VERIPIXEL: CODE-AS-VERIFICATION FOR SELF-IMPROVING PIXEL PREDICTION

Anonymous authors
Paper under double-blind review

ABSTRACT

Generative foundation models attract increasing attention for pixel prediction, as the image-to-image formulation offers the potential to unify image generation and pixel-level understanding tasks. However, existing prompt-based approaches typically rely on fixed, handcrafted prompts for single-round inference. They struggle with the limited semantic understanding capability of generative models, leaving a substantial performance gap relative to tuning-based methods. To this end, we propose VeriPixel, a verification-driven agentic framework for self-improving pixel prediction, which uses verification feedback to progressively enrich generation guidance with semantic information missing from the generative model. First, to provide reliable feedback for self-improvement, we propose code-as-verification, which uses a vision-language model (VLM) to generate executable code, transforming the vanilla single-pass VLM-as-a-judge into an iterative perception-execution-observation process. Instead of image-level VLM judgment, it provides iterative, pixel-grounded verification process. Nevertheless, producing reliable feedback through code execution requires an effective verification strategy. Therefore, instead of a single improvement loop, we propose bootstrapped self-improvement that first uses verifiable feedback from annotated training data to optimize verification strategies offline. With these strategies as instructions, the verifier provides reliable feedback and pixel evidence to bootstrap online self-improvement. Experiments show that VeriPixel improves upon direct prediction by 18.5 cIoU, 11.7 mIoU, and 3.4 dB PSNR across three representative pixel tasks, demonstrating significant gains from self-improvement and achieving competitive performance against tuning-based methods.

1 INTRODUCTION

Pixel prediction tasks aim to produce dense and spatially aligned outputs from input images, encompassing applications such as segmentation (Chen et al., 2017; Liu et al., 2025) and image restoration (Zuo et al., 2025a; Sun et al., 2026). These tasks assign semantic labels (Cheng et al., 2022), estimate scene geometry (Yang et al., 2024b), or recover clean visual content at the pixel level (Zamir et al., 2022), rather than producing image-level labels or textual descriptions. By representing their outputs as images, these diverse tasks can be approached within a unified image-to-image prediction paradigm (Wang et al., 2023), and an increasing number of studies build on this formulation to explore generative foundation models for unified pixel prediction.

Two representative paradigms have emerged within this line of work: **tuning-based** and **prompt-based** methods. Tuning-based methods, such as Vision Banana (Gabeur et al., 2026) and GenCeption (Wang et al., 2026), fine-tune generative models on a mixture of pixel prediction and general-purpose generation data. Despite their effectiveness, frontier model weights and source generation data are often **unavailable**, and extending these methods to new tasks typically requires further fine-tuning (Gabeur et al., 2026; Han et al., 2026; Liu et al., 2026), which limits their practical applicability. Prompt-based methods instead keep model parameters fixed, and generate pixel predictions using handcrafted task-specific prompts (Zuo et al., 2025a; Sun et al., 2026; Liu et al., 2026; Ramachandran et al., 2025). Nevertheless, these methods rely on fixed prompt templates, struggle with the **limited** semantic understanding capability of generative models, leaving a substantial performance gap relative to tuning-based methods (Zuo et al., 2025a; Liu et al., 2026). To address

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.

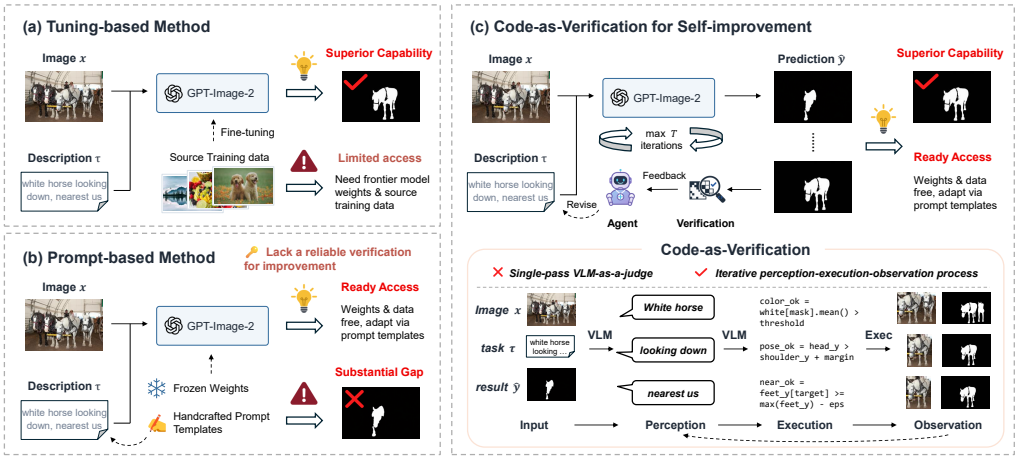

Figure 1: Comparison of paradigms for unified pixel prediction. (a) Tuning-based methods achieve strong performance but are often inaccessible. (b) Prompt-based methods avoid these requirements, but exhibit a substantial performance drop. (c) VeriPixel retains the accessibility, and approaching tuning-based performance through code-as-verification driven self-improvement.

these limitations, as illustrated in Fig. 1, we propose an agentic framework that shifts single-round prompt-based inference to progressive prompt refinement to bridge the semantic gap.

Our central insight is that *self-improvement in pixel prediction requires reliable verification, and* ***code-as-verification*** *provides such verification through an iterative perception-execution-observation process*. First, effective self-improvement depends on reliable feedback, as demonstrated by executable test cases in coding (Zhang et al., 2026; Lee et al., 2026). For pixel prediction, however, ground-truth annotations are unavailable at test time, making feedback a core challenge. To address this challenge, beyond the vanilla vision-language model (VLM) as a judge, we propose utilizing VLM to generate codes for pixel processing and metric computation, which provides informative intermediate observations and quantitative evidence. Moreover, the quantitative evidence links these observations to specific regions, error types, and unmet task requirements, providing more accurate and informative error attribution that helps self-improvement.

Building on this insight, we propose to use code-as-**veri**fication for self-improving **pixel** prediction, named **VeriPixel**. VeriPixel centers on **code-as-verification** and employs **bootstrapped self-improvement** to optimize verification strategies offline, then use verification feedback for online prompt refinement. First, code-as-verification grounds VLM judgments in pixel evidence. Rather than vanilla VLM-as-a-judge, the VLM perceives the input images, executes code combining classical image-processing operators to construct and refine a proxy ground truth, and observes multiple visualizations to validate the proxy. Then, the VLM executes code to compute metrics against this proxy, providing quantitative evidence for quality assessment and error attribution. Nevertheless, producing reliable feedback through code execution depends on appropriate verification strategies for proxy construction, metric selection, and thresholds for assessing prediction quality. Therefore, instead of directly applying a improvement loop, we propose a bootstrapped process that first uses verifiable feedback from annotated training data to optimize the verification strategies offline. Next, incorporating the optimized strategies into the VLM instructions, code-as-verification provides reliable feedback to progressively refine the generative foundation model's input prompt.

Our main contributions are summarized as follows:

- We introduce VeriPixel, a verification-driven agentic framework that bridges the semantic understanding gap in prompt-based unified pixel prediction through self-improvement.
- We propose code-as-verification to transforms VLM-as-a-judge into an iterative perception-execution-observation process, and bootstrapped self-improvement to optimize verification offline for online prompt refinement.
- Without tuning, VeriPixel improves upon direct prediction by 18.5 cIoU, 11.7 mIoU, and 3.4 dB PSNR across three representative pixel tasks, demonstrating significant gains from self-improvement and achieving competitive performance against tuning-based methods.

2 Related Work

2.1 Pixel-level Visual Understanding

Pixel-level visual understanding recently shows a clear trend toward task unification. **Specialist models** use dedicated architectures and supervision, including semantic decoders or vision-language alignment for segmentation (Chen et al., 2018; Wang et al., 2022) and reconstruction networks or generative priors for restoration (Liang et al., 2021; Zamir et al., 2022; Wang et al., 2024).

Task-specific methods broaden capabilities within a domain by supporting multiple prediction settings through shared models or agentic frameworks. Promptable segmentation models support diverse requests (Kirillov et al., 2023; Zou et al., 2023; Li et al., 2024; Lai et al., 2024), while SAM 3 Agent iteratively prompts and inspects masks for complex language guidance (Carion et al., 2026). For image restoration, all-in-one models handle multiple degradations using a single network, (Jiang et al., 2024; Potlapalli et al., 2023; Li et al., 2025), and restoration agents compose specialist tools for mixed degradations (Zhu et al., 2025; Zuo et al., 2025b).

Unified pixel prediction instead represents diverse task outputs within a shared image-to-image formulation (Wang et al., 2023). Tuning-based methods, including Vision Banana and SenseNova-Vision, train generative models on mixtures of pixel prediction and auxiliary data (Gabeur et al., 2026; Han et al., 2026). This training enables a single generator to cover diverse tasks but requires model parameters and source pretrainin data. Prompt-based methods use frozen generators through instructions and prompt chains (Liu et al., 2026; Yang et al., 2025; Zuo et al., 2025a; Ramachandran et al., 2025). They support closed-source models without requiring parameters or pretraining data, but show lower segmentation accuracy and structural fidelity (Liu et al., 2026; Yang et al., 2025; Zuo et al., 2025a). VeriPixel retains this accessibility and narrows the performance gap through verification-driven self-improvement.

2.2 Agentic Framework

Agentic frameworks combine tools, scaffolds, and harnesses for multi-step execution. Tools provide external operations such as search and computation (Schick et al., 2023), scaffolds organize reasoning, actions, and observations (Yao et al., 2023; Wu et al., 2024). Harnesses add context management and execution environments (Yang et al., 2024a; Wang et al., 2025; Lee et al., 2026), enabling sustained software interaction in systems such as SWE-agent and OpenHands (Yang et al., 2024a; Wang et al., 2025).

Editable harnesses also enable **r**ecursive **s**elf-**i**mprovement (**RSI**) by making the agent's own tools and workflows objects of optimization. The Darwin Gödel Machine evaluates such modifications for performance gains (Zhang et al., 2026), and recent methods refine agent code and workflows from execution feedback without updating model weights (Hu et al., 2025; Zhang et al., 2025; Lee et al., 2026). These methods shift improvement from model parameters to the systems that organize model capabilities. Reliable verification is therefore essential for establishing whether each revision improves performance (Zhang et al., 2026; Lee et al., 2026). Executable tests provide automated correctness feedback for coding tasks (Yang et al., 2024a; Zhang et al., 2026).

Visual tasks, especially pixel prediction, lack such reliable feedback because their evaluation typically requires ground truth that is unavailable during inference (Wang et al., 2021). VeriPixel addresses this gap through code-as-verification beyond vanilla VLM-as-a-judge.

3 Methodology

3.1 Framework Overview and Task Formulation

For clarity, we first provide an overview of VeriPixel and its task formulation, as shown in Fig. 2. VeriPixel comprises a frozen generative foundation model, an agent for prompt revision, a VLM-based verifier, and a code execution environment [1]. The agent constructs and refines sample-specific generation guidance g, the verifier generates and executes code to assess prediction quality and pro-

[1] We run Python code with OpenCV in SWE-Vision (Chen et al., 2026)

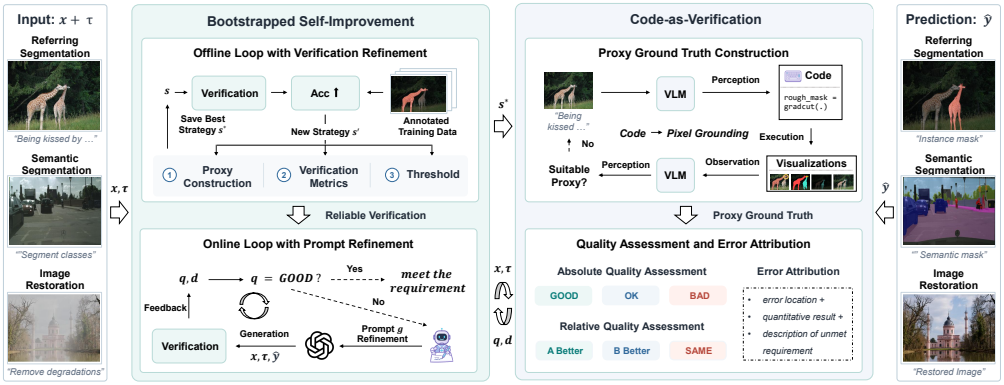

Figure 2: Overview of VeriPixel. Offline refinement selects verification strategies based on annotated training data. Online, code-as-verification provide reliable feedback to enable progressively refine generation prompt.

vide error attribution. All model parameters remain fixed throughout the framework. VeriPixel optimizes a verification strategy s offline and generation guidance g online. Importantly, the framework involves three distinct iterative processes: (1) An inner perception-execution-observation process within each code-as-verification call. (2) An offline loop that optimizes the verification strategy s using already annotated training data. (3) An online loop that progressively refines generation guidance g for each test sample.

For a pixel prediction task, let $x \in \mathbb{R}^{H\times W\times 3}$ denotes an input image and τ denotes its task description. The goal is to generate a spatially aligned output $\hat{y} \in \mathbb{R}^{H\times W\times 3}$, such as a segmentation mask or a restored image. We use a generative foundation model G as a shared prediction backbone across tasks. The agent produces sample-specific guidance g, which is appended to the task description τ to form the generation prompt:

$$\hat{y} = G(x, [\tau; g]) \,, \tag{1}$$

where $[\tau; g]$ denotes the textual concatenation of the task description and generation guidance. During online self-improvement, the agent updates g and keeping x, τ. Let $s = (s_{\text{proxy}}, s_{\text{metric}}, s_{\text{threshold}})$ denote the verification strategy, its components specify how to construct proxy references, select metrics, determine quality thresholds. The strategy is optimized offline and fixed during test-time self-improvement.

3.2 Code-as-Verification

From VLM-as-a-judge to a perception–execution–observation process. Rather than directly producing a judgment from a single-pass visual perception, code-as-verification grounds verification in an iterative perception-execution-observation process, as formulated as:

$$(x, \tau, \hat{y}) \xrightarrow{\text{perception}} c^k \xrightarrow{\text{execution}} \tilde{x}^k \,, \quad \text{observation} \tag{2}$$

where c^k denotes the verification code generated at round k, conditioned the input and observations $\tilde{x}^{k-1}$ from previous rounds. The execution environment runs Python and classical image-processing operators without invoking additional models. During *perception*, the verifier interprets the task requirements and identifies regions or properties requiring inspection. During *execution*, it generates code to process pixels, construct visual evidence, and compute measurements. During *observation*, it checks the resulting masks, overlays, crops, boundaries, or numerical measurements and determines whether further analysis is required. The process terminates when sufficient evidence has been collected or the verification-round budget is reached.

Proxy ground-truth construction. Since test-time ground truth is unavailable, the verifier constructs a proxy reference $\tilde{y}$ from observable image evidence. Guided by the optimized strategy s, proxy construction may use only the input image or additionally condition on the current prediction:

$$\tilde{y} = \text{Exec}\,(c_r; x, \tau, [\iota_s \hat{y}]) \,, \qquad \iota_s \in \{0, 1\}, \tag{3}$$

where Exec denotes code execution, c_r is the proxy-construction code, and ι_s specifies whether prediction conditioning is needed by the verification strategy. When prediction conditioning is used, the prediction serves only as an initialization or region proposal, the verifier validates and revises the resulting proxy against evidence from the original input image. This validation reduces the risk of directly propagating prediction errors into the proxy.

For segmentation, proxy construction usually uses color cues, spatial relationships, GrabCut, thresholding, and coordinate-based masks. For restoration, classical enhancement operators produce references that preserve object structure and scene layout. The specific inputs and operators are selected during offline verification optimization rather than fixed uniformly across tasks.

Quality assessment and error attribution. After constructing the proxy, to provide quantitative feedback, the verifier generates metric-computation code:

$$\mathbf{m} = \mathrm{Exec}\left(c_m; \mathcal{Y}, \tilde{y}, \tau\right), \tag{4}$$

where c_m denotes the generated metric code and $\mathbf{m}$ is the resulting metric values. Segmentation measurements usually include region overlap, precision, and boundary agreement, while restoration measurements may include PSNR, multi-scale SSIM, and localized structural comparisons. $\mathcal{Y}$ contains either a single prediction for absolute quality assessment or two candidates for joint absolute assessment and relative quality comparison. For pairwise verification, both candidates are assessed against the same proxy reference. Next, under strategy s, the verifier assesses prediction quality and attributes errors using the visual observations and metric values:

$$(q, d) = \mathcal{J}_s\left(x, \tau, \mathcal{Y}, \tilde{y}, \mathbf{m}\right), \tag{5}$$

where q is a quality judgment and d is an error attribution. For absolute assessment, the verifier distinguishes three levels of prediction quality by assigning $q \in \{\textit{BAD}, \textit{OK}, \textit{GOOD}\}$. For pairwise assessment, $q \in \{\textit{A Better}, \textit{B Better}, \textit{SAME}\}$. The attribution d describes error locations, error types, and unmet task requirements, providing actionable feedback for prompt refinement.

3.3 Bootstrapped Self-Improvement

Most existing self-improvement frameworks focus on executable tasks such as coding, whereas verification for pixel prediction is particularly challenging because ground truth is unavailable at test time. Therefore, instead of directly applying a self-improvement loop, we transform it into a bootstrapped verification-driven process: First, an offline loop utilizes verifiable feedback from already annotated training data to summarize effective verification strategies. Next, with the guidance of strategies, the code-as-verification provides reliable and visually grounded evidence to enable prompt-level self-improvement through an online loop.

Offline Loop with Verification Refinement. To discover effective verification strategies, we construct quality assessment examples from already annotated training data. For each training image, prompt variations produce candidate predictions spanning different quality levels. Ground-truth annotations provide absolute quality labels and pairwise preference labels, forming a verification evaluation set:

$$\mathcal{D}_{\mathrm{ver}} = \{(z_j, a_j)\}_{j=1}^{N}, \qquad a_j = \mathcal{L}_{\tau_j}(\mathcal{Y}_j, y_j^{\mathrm{gt}}), \tag{6}$$

where $\mathcal{Y}_j$ contains one (*absolute quality*) or two predictions (*relative quality*), $z_j = (x_j, \tau_j, \mathcal{Y}_j)$ is a verification input, a_j is a quality assessment label, and $\mathcal{L}_{\tau_j}$ converts ground-truth into quality labels. To prevent ground-truth leakage, these annotations are used only to construct evaluation targets and assess verification accuracy, they are withheld from the verifier. Let s denote a verification strategy specifying proxy construction, metric selection, and code patterns. The offline loop optimizes s to maximize judgment accuracy:

$$s^{\star} = \underset{s \in \mathcal{S}_{\mathrm{explored}}}{\arg\max} \frac{1}{|\mathcal{D}_{\mathrm{ver}}|} \sum_{(z,a) \in \mathcal{D}_{\mathrm{ver}}} \mathbf{1}[q_s(z) = a], \tag{7}$$

where $q_s(z)$ is the judgment returned by the verification process in Eq. 5, $\mathcal{S}_{\mathrm{explored}}$ contains the agent self-proposed evaluated strategies. Candidate strategies are refined using overall and task-wise judgment accuracy from cross-validation, with the best-performing strategy retained for test-time use. This process optimizes the verification strategy without fine-tuning its underlying VLM.

The retained strategy is encoded as verification instructions and reusable code patterns, rather than a single script applied unchanged to every image. During inference, the VLM instantiates these procedures for the current sample and interprets the returned observations.

Online Loop with Prompt Refinement. At test time, the agent writes generation instructions to generative model based on the task description τ and sample-specific generation guidance g. To enable efficient and effective prompt refinement, we structure the guidance g at four possible granularities:

$$g = \{g_\ell\}_{\ell \in \mathcal{K}}, \qquad \mathcal{K} \subseteq \{\text{image}, \text{region}, \text{instance}, \text{class}\}, \tag{8}$$

where $\mathcal{K}$ specifies the selected granularities and g_ℓ describes the optimized guidance at granularity ℓ. These prompts may clarify target identity, distinguish classes, or specify local corrections. At iteration t, $\hat{y}_t^{\text{best}}$ denotes the best prediction retained from the preceding iterations, and g_t^{best} denotes the guidance that produced it. The verifier evaluates $\hat{y}_t^{\text{best}}$ under the fixed strategy $s^\star$ and returns a quality judgment q_t and error attribution d_t. The agent uses this feedback to revise g_t^{best} into g'_{t+1} and generates the next candidate as

$$\hat{y}'_{t+1} = G\big(x, [\tau; g'_{t+1}]\big), \tag{9}$$

The revision may add or replace guidance granularities and their associated instructions. Each generation uses the original input image and the revised prompt, rather than treating the previous prediction as an image-editing input. The verifier then compares the new candidate $\hat{y}'_{t+1}$ with the retained best prediction $\hat{y}_t^{\text{best}}$. Let q_t^{pair} denote the preferred candidate returned by this comparison. VeriPixel updates the prediction and its associated guidance together:

$$(\hat{y}_{t+1}^{\text{best}}, g_{t+1}^{\text{best}}) = \begin{cases} (\hat{y}'_{t+1}, g'_{t+1}), & q_t^{\text{pair}} = \text{new}, \\ (\hat{y}_t^{\text{best}}, g_t^{\text{best}}), & \text{otherwise}. \end{cases} \tag{10}$$

Subsequent revisions proceed from this retained state until the verifier assigns the prediction an absolute judgment of *GOOD*, or a maximum budget of T iterations is reached. The retained prediction is returned at termination, and no further refinement calls are made for samples that have already received a *GOOD* judgment. This stopping decision uses the verifier's assessment rather than ground-truth, therefore *GOOD* denotes assessed adequacy, not guaranteed correctness. Again, it is crucial to ensure the reliability of verification, which is discussed in Sec. 4.4.

4 Experiments

4.1 Experimental Setup

Datasets and metrics. We evaluate VeriPixel on three typical pixel prediction tasks: referring image segmentation (R-Seg), semantic segmentation (S-Seg), and image restoration (IR). For R-Seg, we evaluate on the validation splits of RefCOCO, RefCOCO+ (Yu et al., 2016) (UNC), and RefCOCOg (Mao et al., 2016) (UMD), using cIoU as metric. For S-Seg, we evaluate on Cityscapes validation (Cordts et al., 2016) using mIoU over 19 classes. For IR, we evaluate on the FoundIR test sets (Li et al., 2025) with 20 real-world degradation types, using full-reference metrics (PSNR, SSIM, LPIPS) and no-reference metrics (CLIP-IQA, MANIQA, MUSIQ).

Default models. By default, the prediction agent uses GPT-5.6-Sol within Codex CLI. The verifier uses GPT-5.5 to interact directly with a Python execution container (Chen et al., 2026), without a separate agent harness. Reasoning effort is set to *high* for both VLMs. GPT-Image-2 generates predictions with quality set to *low*.

Offline Data. We illustrate the details of data in offline verification loop. **Source data.** We use annotated training samples to refine and evaluate offline verifications. Specifically, 20 randomly sampled images from each R-Seg dataset, 100 Cityscapes training images with at least 40 occurrences of each of the 19 classes, and 5 randomly sampled images per degradation across 20 types for IR. **Refinement.** For each sample, we generate 10 predictions with contrasting prompt instructions: 5 with "prioritize image/mask quality without requiring pixel alignment" and 5 with "strictly preserve pixel alignment," to induce quality variation. **Held-out evaluation data.** For offline verification capability in Sec. 4.4, we use 300 additional source samples per task, disjoint from the training data, and generate 10 predictions per sample using the same protocol.

Table 1: Main results on three pixel prediction tasks. Ref., Ref.+, and Ref.g denote RefCOCO, RefCOCO+, and RefCOCOg. ⋆ denotes results reported by ProbeGen (Di et al., 2026) on 500 randomly sampled images per benchmark. Note that Nano-IR results are obtained using GPT-Image-2 with the SF3 prompt (Sun et al., 2026).

Method	Ref.	Ref.+	Ref.g
Specialist Zero-shot Methods			
HybridGL	41.8	35.7	42.5
LH-LLaVA-1.5-13B	76.1	64.1	67.7
SAM3 + Gemini	75.5	67.3	73.4
Prompt-based Unified Methods			
Nano Banana Pro⋆	63.7	53.6	67.1
Nano Banana 2⋆	63.3	53.0	63.9
GPT-Image-2	63.0	50.2	55.7
Tuning-based Unified Method			
Vision Banana	—	—	73.8
VeriPixel (ours)	**76.4**	**69.9**	**74.2**

Method	Cityscapes
Specialist Zero-shot Methods	
OpenSeeD	47.8
X-Decoder	52.0
SAM3	65.2
Prompt-based Unified Methods	
Qwen-Image-Edit	25.7
LongCat-Image-Edit	17.1
GPT-Image-2	52.9
Tuning-based Unified Method	
Vision Banana	**69.9**
VeriPixel (ours)	64.6

Method	PSNR↑	SSIM↑	LPIPS↓	CLIP-IQA↑	MANIQA↑	MUSIQ↑
Specialist Tuning-based Methods						
AirNet	21.5	0.71	0.36	0.33	0.24	38.5
PromptIR	21.8	0.71	0.33	0.34	0.26	39.0
InstructIR	22.2	0.73	0.31	0.38	0.27	41.9
AutoDIR	22.5	0.73	0.29	0.34	0.30	51.2
AgenticIR	21.7	0.71	0.27	0.35	0.28	49.4
4KAgent	22.2	0.72	0.28	0.35	0.31	50.7
Prompt-based Unified Methods						
GPT-Image-2	19.5	0.60	0.27	**0.53**	**0.43**	**67.2**
Nano-IR	20.4	0.62	0.26	0.49	0.41	61.3
VeriPixel (ours)	**22.9**	**0.75**	**0.25**	<u>0.51</u>	<u>0.41</u>	<u>64.9</u>

Inference settings. For each task, we optimize all samples together for at most 15 refinement rounds. When the verifier returns an absolute quality judgment of *GOOD* for a sample, that sample has reached the optimization target and is no longer refined.

4.2 Main Results

To evaluate the pixel prediction capability, we compare VeriPixel with four types of methods. Specialist zero-shot methods are pretrained for related tasks and transferred to new evaluation protocols without additional fine-tuning. For IR, specialist methods are trained on the same evaluation protocol, therefore grouped under specialist tuning-based methods. Prompt-based unified methods perform image-to-image pixel prediction with fixed generative foundation models. Tuning-based unified methods adapt generative foundation models to pixel prediction through parameter updates.

According to the results shown in Tab. 1, we have three main observations. 1) **Without fine-tuning, VeriPixel remains competitive with all tuning-based segmentation methods.** It achieves the best performance across three referring segmentation datasets, particularly exceeds Vision Banana by 0.4% cIoU on RefCOCOg. On Cityscapes, VeriPixel exhibits a modest performance gap relative to SAM3 and Vision Banana due to small categories such as pole and traffic light. 2) **VeriPixel is competitive with tuning-based restoration methods in fidelity and superior in perceptual quality.** On full-reference metrics, VeriPixel is competitive with all tuning-based methods. Although iterative refinement toward higher fidelity incurs a small perceptual-quality trade-off relative to direct prediction, VeriPixel still clearly outperforms all tuning-based methods on no-reference metrics, exceeding the strongest baseline by 0.13 CLIP-IQA, 0.10 MANIQA, and 13.7 MUSIQ. 3) **Self-improvement improves segmentation accuracy and restoration fidelity.** Relative to GPT-Image-2, VeriPixel improves RefCOCOg cIoU by 18.5, Cityscapes mIoU by 11.7, and FoundIR PSNR by 3.4 dB, demonstrating consistent gains across segmentation and restoration tasks.

4.3 Self-Improvement and Efficiency

We analyze how different VLM-harness configurations affect the self-improvement process on RefCOCOg, FoundIR, and Cityscapes. Specifically, we use Codex CLI with GPT-6 Astra and GPT-5.6 Sol, Gemini CLI with Gemini 3.7 Flash and Gemini 3.8 Flash, and DeepSeek Harness with Kimi K3 and DeepSeek-V4.1-Flash. The generative foundation model remains fixed to GPT-Image-2, and verification follows the settings in Sec. 4.1. We presents the curves of process and corresponding qualitative examples.

According to the results shown in Fig. 3, we have the following four observations. 1) **VLM-harness configurations mainly affect convergence efficiency**. We can observe that different configurations finally reach comparable performance by the end of the 15-iteration budget. Only on Cityscapes, GPT-6 Astra and Kimi-K3 exhibit clear advantages. 2) **GPT-6 Astra and Kimi K3 generally improve predictions faster**. Their configurations achieve substantial gains within the first few iterations, whereas Gemini-based configurations typically require more iterations to reach similar performance. 3) **Explicit semantic guidance is crucial for generation**. The examples show how the agent identifies what the generation model misunderstands, clarifies prediction targets, and adds

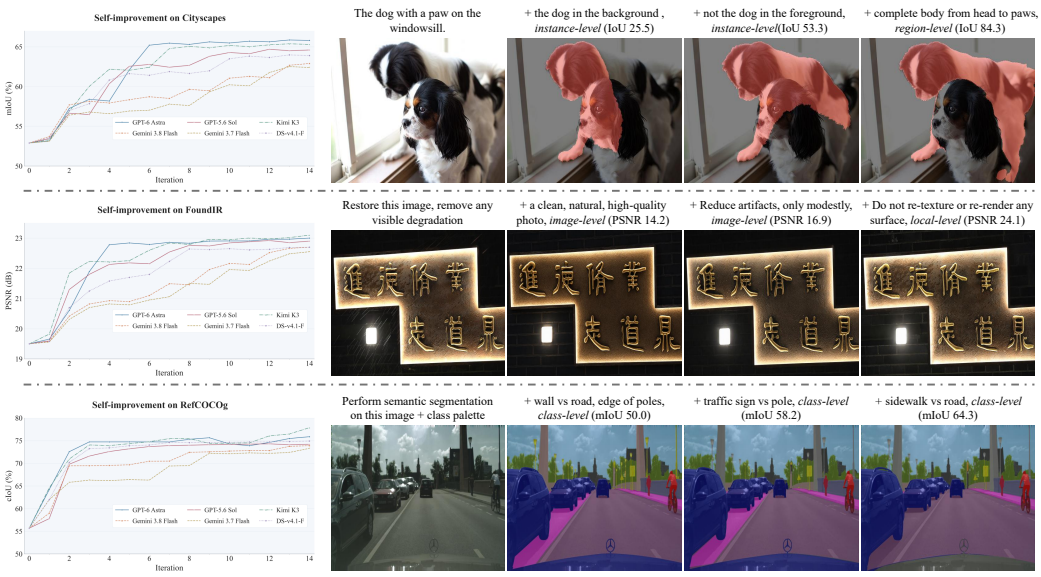

Figure 3: Self-improvement processes and qualitative cases across R-Seg, IR, and S-Seg. On the left, we compare self-improvement processes using different VLMs as agents. DS and F denote DeepSeek and Flash. On the right, qualitative cases illustrate this process for each of the three tasks. "+" denotes crucial improvements of the prompt, with italics indicating its granularity.

guidance based on error attribution to provide explicit guidance, preserve image details, and resolve class confusion. 4) **Guidance granularity varies across tasks and iterations**. The examples show guidance from instance-level to region-level in R-Seg, and from image-level to local-level in IR, and S-Seg uses class-level guidance to resolve successive category confusions.

4.4 Verification Analysis

We evaluate verification capability on RefCOCOg using the held-out predictions described in Sec. 4.1, measuring accuracy against ground-truth derived absolute and relative quality labels. We compare GPT-5.6 Sol, GPT-5.5, Gemini 3.8 Flash, Gemini 3.7 Flash, Kimi K3, and DeepSeek-V4.1-Flash as verifiers, each interacting directly with the Python execution environment. For GPT-5.5 and Gemini 3.7 Flash, we further examine the effects of removing offline optimization or code execution, with the latter reducing verification to VLM-as-a-judge. For the online comparison, we retain the default agent (GPT-5.6 Sol) and GPT-Image-2, and vary only the verifier, comparing GPT-5.5, DeepSeek-V4.1-Flash, the two GPT-5.5 ablations, and an oracle that uses ground truth.

Offline verification capability. The left and middle panels of Fig. 4 provide three observations. 1) **Most VLMs achieve comparable verification accuracy**. GPT and Gemini models differ by at most 2.1% in either absolute or relative accuracy, whereas Kimi K3 and DeepSeek-V4.1-Flash achieve lower accuracy, with gaps of up to 11.1%. 2) **Verification settings have a substantially larger effect on verification than VLM choice**. For GPT-5.5, removing offline optimization reduces absolute and relative accuracy by 17.3% and 16.1%, while reverting to VLM-as-a-judge (without code) causes drops of at least 33.2%. 3) **Gemini 3.7 Flash performs better under both ablations**. Its advantage over GPT-5.5 is at least 1.3% without offline optimization and at least 7.7% without code execution, with the larger advantage in direct visual judgments potentially reflecting stronger visual perception capability.

Effect of verification reliability on self-improvement. The right panel of Fig. 4 provides two further observations. 1) **More accurate verification enables better self-improvement**. The loop achieves higher final cIoU with GPT-5.5 than with DeepSeek-V4.1-Flash and improves further with the oracle, indicating that more reliable verification offers additional gains without changing the agent or generator. 2) **Additional iterations alone do not ensure continued improvement**. Without offline optimization or code-as-verification, performance becomes unstable and may deteriorate with further iterations. In contrast, our method maintains its performance gains across iterations, demonstrating the importance of reliable verification for sustained self-improvement.

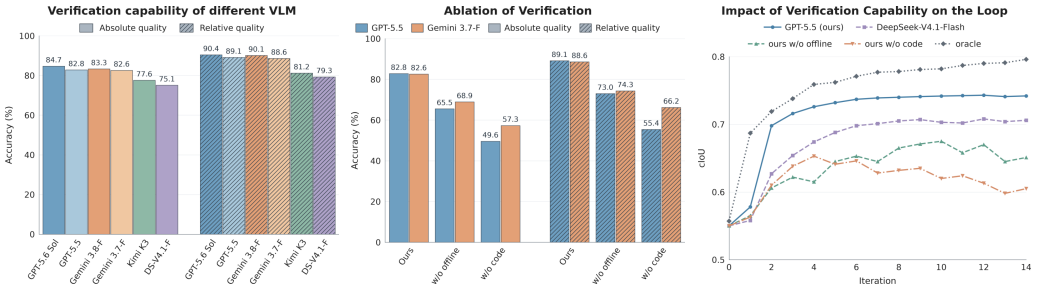

Figure 4: Evaluation of verification reliability and its impact on self-improvement on RefCOCOg. The left panel compares verification performance across VLMs. The middle panel evaluates variants without offline optimization or without code execution (VLM-as-a-judge). The right panel compares self-improvement under different verifiers, including an oracle using ground truth.

Table 2: Time and token usage. Note that the verifier reuses the preceding round's proxy ground truth. We use Time denotes the first round and Time† the average per round for detailed discussion. G-Input/G-Output counts GPT-Image-2 tokens.

Task	Offline Verification Refinement			Online Verification Inference					Online Self-improvement				
	Time (min)	Input (M)	Output (M)	Time (s)	Time† (s)	Rounds	Input (K)	Output (K)	Time (min)	G-Input (K)	G-Output (K)	Input (K)	Output (K)
R-Seg	67.15	20.08	0.14	2.64	1.01	7.85	40.97	8.71	1.53	7.85	1.33	573.89	1.12
S-Seg	75.22	24.39	0.18	4.56	1.29	5.20	35.66	7.31	3.31	16.85	8.53	575.72	1.89
IR	53.96	21.74	0.13	0.69	0.57	6.39	23.53	4.56	0.60	4.53	1.79	145.59	0.32

4.5 Cost Discussion

We discuss the cost of our VeriPixel. Offline verification refinement reports the **total cost over all refinement data per task**. Online verification inference and online self-improvement report **per-sample averages measured under actual concurrent execution**, with 48 concurrent verification requests and 12 concurrent image generation requests. Note that starting from the second round, the verifier reuses the preceding round's proxy ground truth rather than reconstructing it, significantly reducing time and token costs.

Based on the results in Tab. 2, we have three observations. 1) **Task difficulty influence online costs more than image resolution.** Despite the high-resolution FoundIR inputs, IR requires less time and fewer VLM tokens than either segmentation task. S-Seg is more challenging than R-Seg because it requires refining multiple semantic classes rather than a single instance, and incurs higher time and agent token usage. 2) **Reusing proxy ground truth significantly reduces verification overhead.** For S-Seg, average per-round verification time is 1.29 seconds, approximately 72% lower than the first-round time of 4.56 seconds, demonstrating that reusing proxy ground truth refined in the previous round can substantially reduce verification overhead. 3) **Self-improvement uses substantially less than the 15-round budget on average.** The mean is 5.38 image generation calls per sample, including initial generation and samples that never receive *GOOD*.

5 Conclusion

In this paper, we present VeriPixel, a verification-driven agentic framework that addresses the limited semantic understanding of frozen generative foundation models by replacing single-round pixel prediction with progressive prompt self-improvement. To provide reliable feedback, we propose code-as-verification to ground VLM judgments in an perception-execution-observation process, producing pixel-level measurements and localized error attribution for prompt refinement. Further, to ensure the effectiveness of verification strategy, bootstrapped self-improvement first optimizes verification strategies using annotated training data offline, and then fixes these strategies at test time to guide sample-specific optimization. Across referring segmentation, semantic segmentation, and image restoration, VeriPixel improves direct GPT-Image-2 predictions by 11.7 mIoU on Cityscapes and 3.4 dB PSNR on FoundIR, remains competitive with tuning-based segmentation methods, and outperforms tuning-based restoration baselines on all no-reference metrics.

ETHICS STATEMENT

This work uses existing RefCOCO/+/g, Cityscapes, and FoundIR datasets under their respective terms, we recruit no human participants, collect no new data, and do not seek to identify individuals. VeriPixel may inherit biases, performance disparities, and privacy risks from these datasets and its underlying models, while segmentation and restoration capabilities could be misused for unauthorized surveillance or deceptive image manipulation. Its iterative verifier introduces further risks: incorrect proxies, metrics, or thresholds may amplify prediction errors rather than guarantee correctness, and model-generated code may pose security risks. We therefore execute generated code in an isolated environment and recommend preserving inputs and provenance, obtaining authorization for sensitive data, and retaining human oversight in high-stakes settings. We discourage surveillance, deceptive editing, and automated decisions that materially affect individuals or communities.

REPRODUCIBILITY STATEMENT

Our code implementation, including the agentic framework, verification procedures, and evaluation scripts, will be publicly released upon acceptance to facilitate further research and community adoption. Implementation details and experimental protocols are documented in the main text and appendix to support reproducibility and independent verification.

AI USE STATEMENT

AI tools are used to assist with manuscript drafting, language editing, organization of the paper, title and keyword suggestions, and code implementation. The authors review and test all AI-assisted code for correctness.

REFERENCES

Nicolas Carion, Laura Gustafson, Yuan-Ting Hu, Shoubhik Debnath, Ronghang Hu, Didac Suris Coll-Vinent, Chaitanya Ryali, Kalyan Vasudev Alwala, Haitham Khedr, Andrew Huang, Jie Lei, Tengyu Ma, Baishan Guo, Arpit Kalla, Markus Marks, Joseph Greer, Meng Wang, Peize Sun, Roman Rädle, Triantafyllos Afouras, Effrosyni Mavroudi, Katherine Xu, Tsung-Han Wu, Yu Zhou, Liliane Momeni, Rishi Hazra, Shuangrui Ding, Sagar Vaze, Francois Porcher, Feng Li, Siyuan Li, Aishwarya Kamath, Ho Kei Cheng, Piotr Dollár, Nikhila Ravi, Kate Saenko, Pengchuan Zhang, and Christoph Feichtenhofer. SAM 3: Segment anything with concepts. In *ICLR*, 2026.

Liang Chen, Haozhe Zhao, Tianyi Ma, and Kuan Li. SWE-Vision: A minimal agent for advancing visual intelligence, 2026. URL `https://unipat.ai/blog/SWE-Vision`.

Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE transactions on pattern analysis and machine intelligence*, 40(4): 834–848, 2017.

Liang-Chieh Chen, Yukun Zhu, George Papandreou, Florian Schroff, and Hartwig Adam. Encoder-decoder with atrous separable convolution for semantic image segmentation. In *ECCV*, pp. 801–818, 2018.

Bowen Cheng, Ishan Misra, Alexander G Schwing, Alexander Kirillov, and Rohit Girdhar. Masked-attention mask transformer for universal image segmentation. In *CVPR*, pp. 1290–1299, 2022.

Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The Cityscapes dataset for semantic urban scene understanding. In *CVPR*, pp. 3213–3223, 2016.

Shangzhe Di, Zhaokai Wang, and Weidi Xie. Are image generators zero-shot perceivers? a rigorous evaluation. *arXiv preprint arXiv:2609.07884*, 2026.

Valentin Gabeur, Shangbang Long, Songyou Peng, Paul Voigtlaender, Shuyang Sun, Yanan Bao, Karen Truong, Zhicheng Wang, Wenlei Zhou, Jonathan T Barron, et al. Image generators are generalist vision learners. *arXiv preprint arXiv:2604.20329*, 2026.

Xiaoyang Han, Jianhua Li, Kewang Deng, Zukai Chen, Xuanke Shi, Sihan Wang, Boxuan Li, Linyan Wang, Siyi Xie, Xin You, et al. Vision as unified multimodal generation. *arXiv preprint arXiv:2607.06560*, 2026.

Shengran Hu, Cong Lu, and Jeff Clune. Automated design of agentic systems. In *ICLR*, 2025.

Yitong Jiang, Zhaoyang Zhang, Tianfan Xue, and Jinwei Gu. Autodir: Automatic all-in-one image restoration with latent diffusion. In *ECCV*, pp. 340–359. Springer, 2024.

Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C Berg, Wan-Yen Lo, et al. Segment anything. In *ICCV*, pp. 4015–4026, 2023.

Xin Lai, Zhuotao Tian, Yukang Chen, Yanwei Li, Yuhui Yuan, Shu Liu, and Jiaya Jia. Lisa: Reasoning segmentation via large language model. In *CVPR*, pp. 9579–9589, 2024.

Yoonho Lee, Roshen Nair, Qizheng Zhang, Kangwook Lee, Omar Khattab, and Chelsea Finn. Meta-harness: End-to-end optimization of model harnesses. *arXiv preprint arXiv:2603.28052*, 2026.

Feng Li, Hao Zhang, Peize Sun, Xueyan Zou, Shilong Liu, Chunyuan Li, Jianwei Yang, Lei Zhang, and Jianfeng Gao. Segment and recognize anything at any granularity. In *ECCV*, 2024.

Hao Li, Xiang Chen, Jiangxin Dong, Jinhui Tang, and Jinshan Pan. Foundir: Unleashing million-scale training data to advance foundation models for image restoration. In *ICCV*, pp. 12626–12636, 2025.

Jingyun Liang, Jiezhang Cao, Guolei Sun, Kai Zhang, Luc Van Gool, and Radu Timofte. SwinIR: Image restoration using Swin Transformer. In *ICCV Workshops*, pp. 1833–1844, 2021.

Wei Liu, Jiaxin Lin, and Rui Chen. Open-source image editing models are zero-shot vision learners. *arXiv preprint arXiv:2605.04566*, 2026.

Yuqi Liu, Bohao Peng, Zhisheng Zhong, Zihao Yue, Fanbin Lu, Bei Yu, and Jiaya Jia. Seg-zero: Reasoning-chain guided segmentation via cognitive reinforcement. *arXiv preprint arXiv:2503.06520*, 2025.

Junhua Mao, Jonathan Huang, Alexander Toshev, Oana Camburu, Alan L. Yuille, and Kevin Murphy. Generation and comprehension of unambiguous object descriptions. In *CVPR*, pp. 11–20, 2016.

Vaishnav Potlapalli, Syed Waqas Zamir, Salman Khan, and Fahad Khan. Promptir: Prompting for all-in-one blind image restoration. In *37th Conference on Neural Information Processing Systems (NeurIPS), New Orleans, LA, dec 10-16, 2023*. NeurIPS, 2023.

Rahul Ramachandran, Ali Garjani, Roman Bachmann, Andrei Atanov, Oğuzhan Fatih Kar, and Amir Zamir. How well does gpt-4o understand vision? evaluating multimodal foundation models on standard computer vision tasks. *arXiv preprint arXiv:2507.01955*, 2025.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessí, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. In *NeurIPS*, volume 36, 2023. doi: 10.52202/075280-2997.

Weixiong Sun, Xiang Yin, and Chao Dong. Can nano banana 2 replace traditional image restoration models? an evaluation of its performance on image restoration tasks. In *CVPR-Workshop*, pp. 4581–4590, 2026.

Dequan Wang, Evan Shelhamer, Shaoteng Liu, Bruno Olshausen, and Trevor Darrell. Tent: Fully test-time adaptation by entropy minimization. In *ICLR*, 2021.

Jianyi Wang, Zongsheng Yue, Shangchen Zhou, Kelvin C. K. Chan, and Chen Change Loy. Exploiting diffusion prior for real-world image super-resolution. *International Journal of Computer Vision*, 132(12):5929–5949, 2024. doi: 10.1007/s11263-024-02168-7.

Letian Wang, Chuhan Zhang, Rishabh Kabra, Jasper Uijlings, Steven Waslander, Andrew Zisserman, Joao Carreira, Kaiming He, Misha Andriluka, Eduard Gabriel Bazavan, et al. Video generation models are general-purpose vision learners. *arXiv preprint arXiv:2607.09024*, 2026.

Xingyao Wang, Boxuan Li, Yufan Song, Frank F Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, Hoang Tran, Fuqiang Li, Ren Ma, Mingzhang Zheng, Bill Qian, Daniel Shao, Niklas Muennighoff, Yizhe Zhang, Binyuan Hui, Junyang Lin, Robert Brennan, Hao Peng, Heng Ji, and Graham Neubig. OpenHands: An open platform for AI software developers as generalist agents. In *ICLR*, 2025.

Xinlong Wang, Wen Wang, Yue Cao, Chunhua Shen, and Tiejun Huang. Images speak in images: A generalist Painter for in-context visual learning. In *CVPR*, 2023.

Zhaoqing Wang, Yu Lu, Qiang Li, Xunqiang Tao, Yandong Guo, Mingming Gong, and Tongliang Liu. CRIS: CLIP-driven referring image segmentation. In *CVPR*, pp. 11686–11695, 2022.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, et al. Autogen: Enabling next-gen llm applications via multi-agent conversation. In *ICLR-Workshop*, 2024.

Hao Yang, Yan Yang, Ruikun Zhang, and Liyuan Pan. A preliminary study on gpt-image generation model for image restoration. *arXiv preprint arXiv:2505.05621*, 2025.

John Yang, Carlos E Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. Swe-agent: agent-computer interfaces enable automated software engineering. In *NeurIPS*, pp. 50528–50652, 2024a.

Lihe Yang, Bingyi Kang, Zilong Huang, Xiaogang Xu, Jiashi Feng, and Hengshuang Zhao. Depth anything: Unleashing the power of large-scale unlabeled data. In *CVPR*, pp. 10371–10381, 2024b.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *ICLR*, 2023.

Licheng Yu, Patrick Poirson, Shan Yang, Alexander C. Berg, and Tamara L. Berg. Modeling context in referring expressions. In *ECCV*, 2016.

Syed Waqas Zamir, Aditya Arora, Salman Khan, Munawar Hayat, Fahad Shahbaz Khan, and Ming-Hsuan Yang. Restormer: Efficient transformer for high-resolution image restoration. In *CVPR*, pp. 5728–5739, 2022.

Jenny Zhang, Shengran Hu, Cong Lu, Robert Tjarko Lange, and Jeff Clune. Darwin gödel machine: Open-ended evolution of self-improving agents. In *ICLR*, 2026.

Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xiong-Hui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, Bingnan Zheng, Bang Liu, Yuyu Luo, and Chenglin Wu. AFlow: Automating agentic workflow generation. In *ICLR*, 2025.

Kaiwen Zhu, Jinjin Gu, Zhiyuan You, Yu Qiao, and Chao Dong. An intelligent agentic system for complex image restoration problems. In *ICLR*, pp. 57985–58013, 2025.

Xueyan Zou, Jianwei Yang, Hao Zhang, Feng Li, Linjie Li, Jianfeng Wang, Lijuan Wang, Jianfeng Gao, and Yong Jae Lee. Segment everything everywhere all at once. In *NeurIPS*, volume 36, pp. 19769–19782, 2023.

Jialong Zuo, Haoyou Deng, Hanyu Zhou, Jiaxin Zhu, Yicheng Zhang, Yiwei Zhang, Yongxin Yan, Kaixing Huang, Weisen Chen, Yongtai Deng, et al. Is nano banana pro a low-level vision all-rounder? a comprehensive evaluation on 14 tasks and 40 datasets. *arXiv preprint arXiv:2512.15110*, 2025a.

Yushen Zuo, Qi Zheng, Mingyang Wu, Xinrui Jiang, Renjie Li, Jian Wang, Yide Zhang, Gengchen Mai, Lihong Wang, James Y Zou, Xiaoyu Wang, Ming-Hsuan Yang, and Zhengzhong Tu. 4KAgent: Agentic any image to 4K super-resolution. In *NeurIPS*, volume 38, 2025b. doi: 10.52202/085713-5476.

APPENDIX

A NOTATION SUMMARY

For clarity, we summarize the principal notation used throughout the paper in Tab. 3. The notation is organized according to task formulation and generation, code-as-verification, the offline loop with verification refinement, and the online loop with prompt refinement. Standard evaluation metrics and model names are omitted.

Table 3: Summary of the principal notation used in VeriPixel.

Notation	Meaning
	Task Formulation and Generation
$x \in \mathbb{R}^{H\times W\times 3}$	Input RGB image with height H and width W.
τ	Task description associated with the input image.
G	Frozen generative foundation model used as the shared pixel prediction backbone.
$y^{\mathrm{gt}}, \hat{y}$	Ground-truth output and generated pixel prediction, respectively. Ground truth is available only during offline refinement and evaluation.
$g, [\tau; g]$	Sample-specific generation guidance and its textual concatenation with the task description.
$\mathcal{K}, g_\ell$	Selected guidance granularities and the guidance at granularity $\ell \in \{\mathrm{image}, \mathrm{region}, \mathrm{instance}, \mathrm{class}\}$.
	Code-as-Verification
s	Verification strategy $s = (s_{\mathrm{proxy}}, s_{\mathrm{metric}}, s_{\mathrm{threshold}})$ specifying proxy-reference construction, metric selection, and quality thresholds.
$s^\star$	Verification strategy selected by offline refinement and fixed during online self-improvement.
$k, c^k, \tilde{x}^k$	Verification round, code generated at that round, and the corresponding execution observation.
$\mathrm{Exec}(c; \cdot)$	Code execution operation that runs code c on the supplied inputs.
c_r, c_m	Generated code for proxy-reference construction and metric computation, respectively.
$\tilde{y}, \iota_s$	Proxy reference and the binary indicator specifying whether its construction conditions on the current prediction.
$\mathcal{Y}$	Prediction set containing one candidate for absolute assessment or two candidates for absolute and relative assessment.
$\mathbf{m}$	Pixel-level measurements computed by executing c_m.
$\mathcal{J}_s$	Verification operation that integrates visual observations and measurements under strategy s.
q, d	Quality judgment and localized error attribution returned by the verifier.
	Offline Loop with Verification Refinement
$\mathcal{D}_{\mathrm{ver}}, N$	Verification evaluation set and its number of examples.
z_j, a_j	Verification input and ground-truth-derived quality label for example j.
$\mathcal{L}_{\tau_j}$	Task-dependent labeling function that converts prediction quality against y_j^{gt} into an absolute or relative label.
$\mathcal{S}_{\mathrm{explored}}$	Set of verification strategies evaluated during offline refinement.
$q_s(z), \mathbf{1}[\cdot]$	Judgment produced for input z under strategy s and the indicator function used to compute judgment accuracy.
	Online Loop with Prompt Refinement
t, T	Current online refinement iteration and the maximum refinement budget.
$(\hat{y}_t^{\mathrm{best}}, g_t^{\mathrm{best}})$	Best prediction retained through iteration t and the generation guidance that produced it.
$(\hat{y}'_{t+1}, g'_{t+1})$	Newly generated candidate prediction and its revised guidance.
q_t^{pair}	Pairwise judgment used to select between the retained prediction and the new candidate.

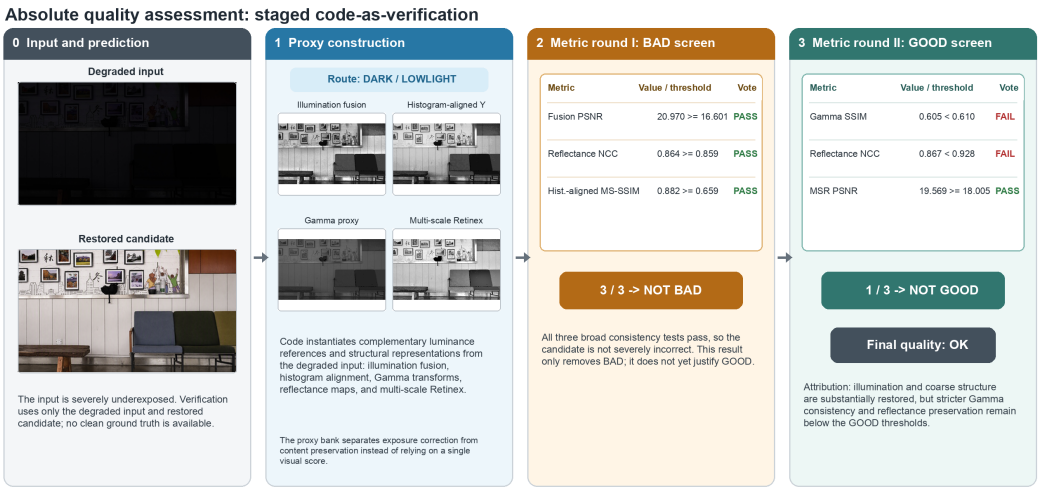

Figure 5: Representative trajectory for absolute quality assessment in image restoration. The verifier routes the sample to a task-specific strategy and constructs complementary proxy representations from the degraded input without prediction conditioning. A broad first metric round excludes *BAD*, and a stricter second round rejects *GOOD*, producing the final label *OK*. The clean target and its metrics are not used in this verification trajectory.

B Case Studies of Code-as-Verification

This section presents representative verification trajectories to illustrate how executable code transforms a single visual judgment into an iterative perception–execution–observation process. Rather than reproducing every tool call, we organize each trajectory around the two functional stages in our framework: proxy-reference construction and metric computation. The intermediate observations retained below are those that materially affect the subsequent verification decision.

B.1 Absolute Quality Assessment

Fig. 5 presents a low-light image-restoration example in which the verifier assigns an absolute quality label to a single restored candidate. The input is severely underexposed, while the candidate recovers most scene content and appears visually plausible. This case illustrates why absolute assessment is implemented as a staged verification process rather than a single similarity measurement.

Proxy-reference construction. The verifier first identifies the sample to the *dark/low-light* image, uses corresponding strategy and executes classical image-processing operators to construct complementary proxy representations. The proxy bank includes illumination-fusion references, guided-reflectance maps, Gamma-corrected references, and multi-scale Retinex references. These proxies capture different requirements of restoration: exposure correction, preservation of illumination-invariant structure, and agreement with plausible enhancement strengths. All proxy references are constructed from the degraded input without using the prediction. The prediction enters only during metric computation, for example, histogram alignment is used as a comparison normalization rather than to refine proxy construction. No clean target image is available to the verifier.

Metric round I: excluding *BAD*. The first round applies a broad quality screen using three complementary measurements. The illumination-fusion PSNR is 20.970 dB against a threshold of 16.601 dB (*offline optimized threshold*), guided-reflectance NCC is 0.864 against 0.859, and histogram-aligned multi-scale SSIM is 0.882 against 0.659. All three measurements pass their thresholds, exceeding the two-vote requirement and yielding the intermediate result *not BAD*. Importantly, this result only rules out a severe restoration failure, it does not distinguish *OK* from *GOOD*.

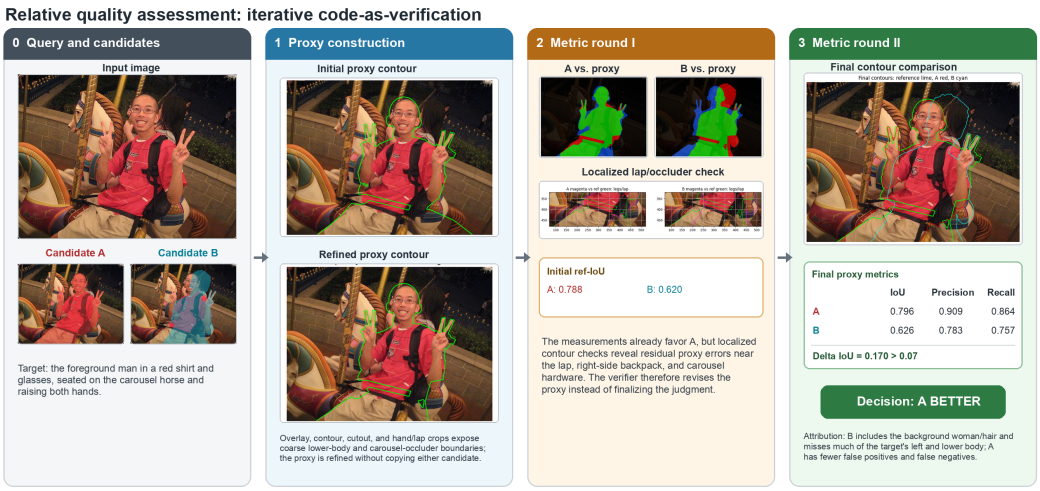

Figure 6: Representative trajectory for relative quality assessment in referring segmentation. The verifier independently constructs and visually validates a proxy reference, performs an initial metric round, and uses localized disagreement evidence to identify remaining proxy errors. After refining the proxy, a second metric round produces the final measurements and localized error attribution. In the disagreement maps, green denotes overlap with the proxy, red denotes candidate-only false positives, and blue denotes proxy regions missed by the candidate.

Metric round II: testing *GOOD*. The second round therefore evaluates the same candidate with stricter proxies and thresholds. Gamma-bank SSIM is 0.605, below its 0.610 threshold, and guided-reflectance NCC is 0.867, below the stricter 0.928 threshold. Only multi-scale-Retinex PSNR passes, with 19.569 dB against 18.005 dB. Consequently, only one of the three measurements votes for *GOOD*, whereas two votes are required. Combining the two rounds gives the final absolute judgment *OK*: the candidate is sufficiently faithful to avoid *BAD*, but the evidence is not strong enough to support *GOOD*.

Error attribution. The complementary measurements indicate that illumination and coarse scene structure are substantially restored, but stricter Gamma consistency and reflectance preservation remain slightly below their quality requirements. Thus, code execution identifies a residual tone and structure mismatch that is difficult to infer from the bright, visually plausible output alone. This case also demonstrates how the verifier reuses the same proxy family with stage-specific thresholds: a measurement can support excluding *BAD* while still failing the stronger evidence required for *GOOD*.

B.2 Relative Quality Assessment

Fig. 6 shows a referring-segmentation example in which the verifier compares two masks for the foreground man in a red shirt and glasses who raises both hands while sitting on a carousel horse. This example is particularly informative because the two candidates have similar overall areas but differ substantially in which person and body regions they cover.

Proxy-reference construction. The verifier first constructs a candidate-independent proxy reference from the input image and referring expression. It executes code to represent the visible head, torso, backpack, hands, and legs while removing the carousel pole, safety bar, and other occluding hardware. The initial proxy is inspected through an alpha overlay, an image-aligned contour, a dimmed-background cutout, and localized crops of the hands and lap. These observations reveal that the lower-body extent and several occluder boundaries are too coarse. The verifier therefore revises the proxy using the raw-image evidence, without copying either candidate boundary.

Table 4: Task-specific proxy construction and measurement strategies. Preferred entries are retained after offline verification refinement, while candidates denote alternative code patterns and measurements considered during strategy exploration. Prediction conditioning refers only to proxy construction, not to metric comparison.

Task	Proxy construction	Measurements
R-Seg	**Prediction conditioning:** No. **Preferred (3):** (1) expression-guided coordinate and part primitives, including polygons and ellipses, (2) ROI-restricted color or appearance thresholding, (3) explicit occluder subtraction followed by morphological and connected-component cleanup. **Candidates (5):** GrabCut, edge-guided region growing, watershed, superpixel merging, convex-hull or contour completion. **Validation:** alpha overlays, contours on the raw image, dim-background cutouts, localized crops, and FP/FN disagreement maps. Inspect target identity and scope, occlusion handling, extra or missing parts, and boundary alignment.	**Preferred (3):** reference IoU, precision–recall, false-positive and false-negative pixel areas. **Candidates (5):** Dice/F1, boundary IoU or boundary F-score, Hausdorff or Chamfer distance, centroid and bounding-box alignment, connected-component and area consistency.
S-Seg	**Prediction conditioning:** No for *train*, *bus*, *truck*, and *wall*, yes for the remaining classes, where the prediction is used only as an initialization or region proposal. **Preferred (3):** (1) class-specific color and texture thresholding, (2) spatial and size priors with connected-component filtering, (3) prediction-seeded region refinement for conditioned classes. **Candidates (5):** GrabCut, k-means or GMM color clustering, edge-guided region growing, watershed, superpixel or contour merging. **Validation:** class-color overlays, per-class contours, component and occupancy maps, and localized boundary crops. Inspect class presence, scene coverage, class confusion, disconnected or thin objects, and boundary leakage.	**Preferred (3):** class-wise IoU/mIoU, class-wise precision–recall, class-presence and coverage accuracy. **Candidates (5):** class-wise Dice/F1, boundary IoU or F-score, per-class area ratio, connected-component consistency, centroid and spatial-occupancy agreement.
IR	**Prediction conditioning:** No. **Preferred (3):** (1) illumination-fusion and Gamma reference banks, (2) guided-reflectance decomposition, (3) multi-scale Retinex and degradation-aware operator banks. **Candidates (5):** histogram equalization or CLAHE, dark-channel or color-constancy correction, bilateral or median denoising, unsharp or Laplacian sharpening, multi-scale smoothing or deblurring banks. **Validation:** proxy/candidate side-by-side views, residual heatmaps, edge or gradient maps, histogram and illumination plots, and localized crops. Inspect scene layout and structure, residual degradation, over-smoothing or over-sharpening, exposure or color shift, and introduced artifacts.	**Preferred (3):** operator-bank PSNR, including illumination-fusion or MSR variants, guided-reflectance NCC, histogram-aligned MS-SSIM or Gamma-bank SSIM. **Candidates (5):** raw PSNR, vanilla SSIM, Laplacian or gradient similarity, census similarity or changed-pixel ratio, brightness and color-histogram distance.

Iterative metric computation. In the first metric round, executable comparison against the refined proxy gives reference IoUs of 0.788 for candidate A and 0.620 for candidate B. Although these measurements already favor A, the false-positive/false-negative maps and localized contour comparisons expose small residual proxy errors around the lap, right-side backpack, and carousel hardware. The verifier consequently returns to proxy construction instead of immediately producing a preference. After the proxy is corrected and revalidated, the second metric round computes IoU, precision, and recall together with localized disagreement maps. Candidate A obtains an IoU of 0.796, precision of 0.909, and recall of 0.864, compared with 0.626, 0.783, and 0.757 for candidate B. The IoU difference of 0.170 exceeds the 0.07 relative-quality threshold, yielding the final judgment *A better*.

Error attribution. The pixel evidence also makes the preference actionable rather than merely assigning a score. Candidate B includes a large non-target region belonging to the background woman and her hair, while omitting substantial portions of the target's left side and lower body. Candidate A still contains minor boundary errors, but has both fewer false positives and fewer false negatives. This trajectory demonstrates that metric execution serves two roles: it distinguishes the candidates and provides observations that can trigger further correction of the verifier's own proxy before the final decision.

C Verification Strategies

This section provides the task-specific verification strategies retained after offline verification refinement. For each task, we summarize proxy-reference construction, metric computation, quality judgment, and error attribution. These strategies are encoded as reusable instructions and code patterns rather than fixed scripts, allowing the verifier to instantiate them for each sample.

Table 5: Reference comparison between GPT-Image-2 and Vision Banana on depth and surface-normal prediction. †Vision Banana results are reported in (Gabeur et al., 2026).

Task	**Metric**	**GPT-Image-2**	**Vision Banana**†
Depth	AbsRel ↓	1.2171	0.081
	δ_1 ↑	17.46%	94.8%
Surface normal	Mean angular error ↓	50.017°	17.778°
	Median angular error ↓	42.213°	8.876°

C.1 Strategy Overview

Tab. 4 summarizes the optimized verification strategies for the three evaluated tasks.

C.2 Cross-Task Analysis and Reuse

Despite task-specific operators and measurements, the optimized strategies share three design principles. First, the proxy representation is matched to the prediction structure. Segmentation requires spatially explicit region and boundary evidence, whereas restoration is better represented by a bank of complementary references that separate exposure correction, structural preservation, and degradation removal. The two trajectories in Sec. B illustrate these alternatives: referring segmentation iteratively corrects a single spatial proxy, while restoration evaluates one candidate against multiple operator-derived proxies with stage-specific thresholds.

Second, proxy construction is coupled with explicit validation rather than treated as a one-shot preprocessing step. Overlays, contours, crops, component statistics, and intermediate metric maps allow the verifier to detect when its own proxy remains inaccurate and to return to code execution. Prediction conditioning is disabled for R-Seg and IR. For S-Seg, it is disabled for *train*, *bus*, *truck*, and *wall*, and enabled for the remaining classes, where the prediction is used only as an initialization or region proposal. Any conditioned proxy must still be checked against evidence from the input image. For relative assessment, all candidates are evaluated against the same validated proxy, preventing the comparison criterion from changing between candidates.

Third, the measurements and decision rules are selected offline and remain fixed during online self-improvement. Segmentation strategies emphasize overlap, precision–recall, and boundary evidence, while restoration combines fidelity, structural, brightness, and texture measurements whose failure modes are complementary. Absolute judgments combine these measurements into *BAD*, *OK*, or *GOOD*, relative judgments additionally apply a preference margin to avoid declaring a winner for negligible differences. In both settings, the verifier returns localized or property-specific error attribution together with the judgment, providing the agent with concrete evidence for prompt refinement.

The proxy constructed in the first online round is reused in subsequent rounds unless its validity checks fail. This reuse reduces execution overhead and, more importantly, provides a stable reference across successive predictions. If new observations reveal a proxy error, the verifier can revise and revalidate it before recomputing the measurements, as demonstrated by the relative-quality case in Fig. 6.

D Limitations

Although VeriPixel improves prompt-based prediction on segmentation and image restoration, its effectiveness remains constrained by the underlying generative model's capability to infer and express the target pixel representation. To examine this boundary, we additionally analyze depth estimation and surface-normal predictions produced directly by GPT-Image-2, respectively. Further , we provide both quantitative and qualitative results for discussion. The comparison is summarized in Tab. 5, while Figs. 7 and 8 present examples selected by error quantiles.

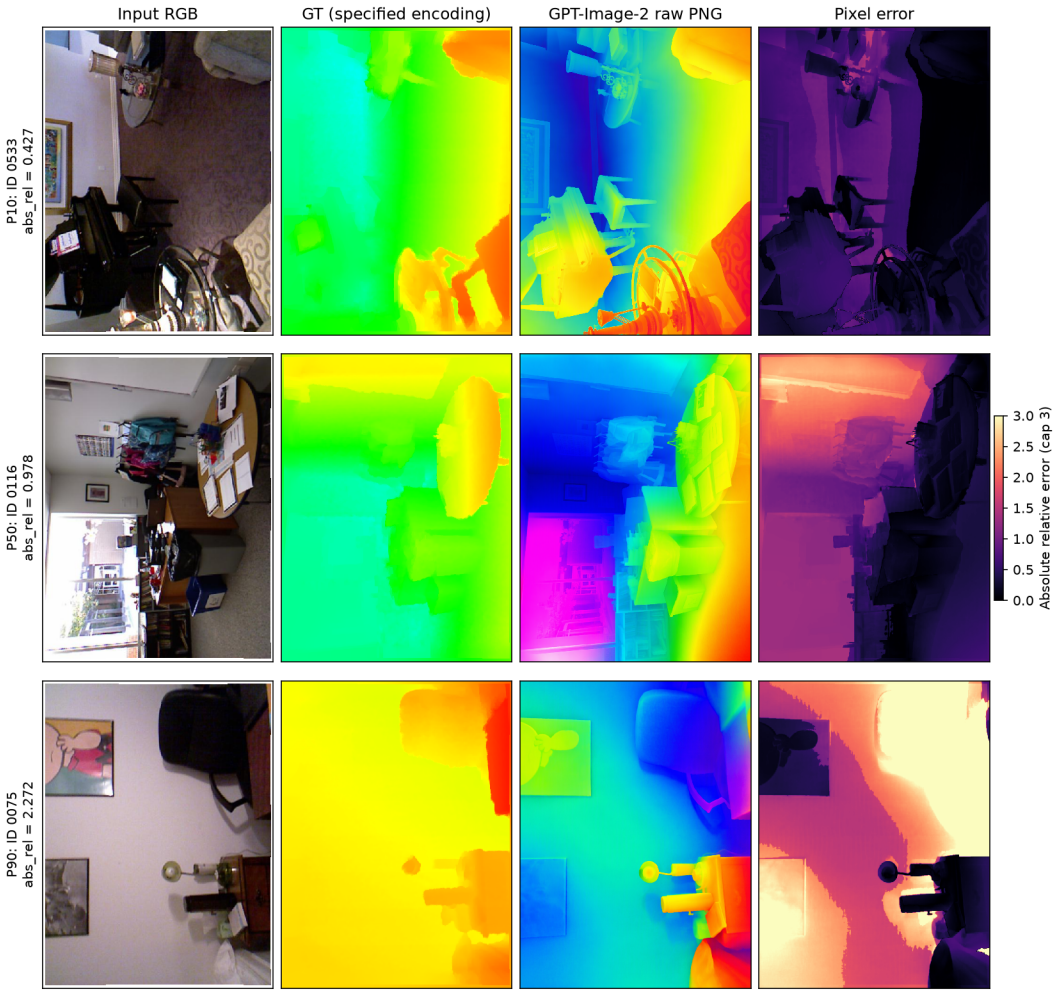

Figure 7: Depth diagnostics selected at the P10, P50, and P90 positions after sorting by the original per-image AbsRel, the examples are not manually selected. Each row shows the input image, ground truth under the specified encoding, the raw GPT-Image-2 output, and the pixel-wise absolute relative error. The outputs remain visually coherent and broadly preserve scene appearance, but the ground truth and error maps reveal incorrect metric ranges and near–far relationships. The figure therefore illustrates that plausible 2D structure does not imply accurate 3D depth. The displayed error is capped at 3 only for visualization.

D.1 Extension from 2D Generation to 3D Geometry

GPT-Image-2 exhibits substantial gaps in metric depth, relative ordering, and local geometric variation. Together with its competitive performance on the 2D tasks evaluated in the main paper, these large reference gaps reveal a substantial weakness of GPT-Image-2 in extending its visual prediction capability to 3D perception. The pairwise analysis further reveals genuine geometric reversals: even for pixel pairs whose ground-truth depths differ by more than 50%, approximately one quarter are ordered incorrectly. Meanwhile, predicted variation is excessive in nearly flat regions but weakened at strong discontinuities, indicating that the magnitude and location of geometric changes are both inaccurate.

Surface-normal prediction exposes a similar limitation. The model's mean angular error is worse than a constant front-facing normal, and neither cross-validated nor per-image oracle axis/sign transformations materially repair the predictions. Although the locally constructed normal labels are

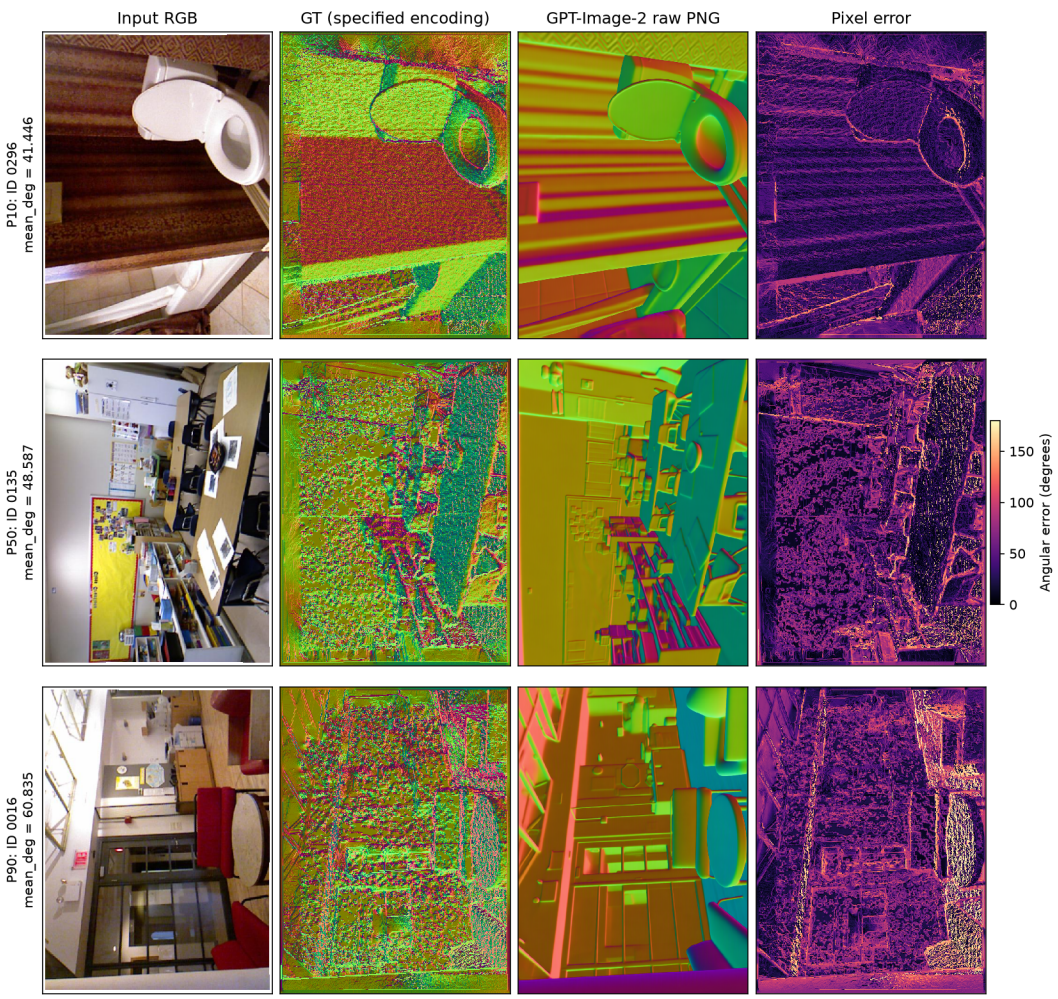

Figure 8: Surface-normal diagnostics selected at the P10, P50, and P90 positions after sorting by the original per-image mean angular error, the examples are not manually selected. Each row shows the input image, locally constructed ground truth under the specified encoding, the raw GPT-Image-2 output, and the directed angular-error map. Although the generated outputs form smooth and visually coherent normal-like images, the distributed angular errors show that they do not recover accurate local surface orientations. The high-frequency sensitivity of the locally constructed ground truth is analyzed separately in the text.

sensitive to high-frequency depth variation, smoothing the labels explains only part of the error and leaves a large residual gap. The examples in Figs. 7 and 8 visually support these aggregate statistics: the outputs often preserve coarse scene appearance while substantially changing depth range, near–far relationships, or local normal structure. Together, the evidence suggests that current prompt-based use of GPT-Image-2 does not reliably extend its 2D generation capability to metric and locally consistent 3D geometry.

D.2 Following Complex Pixel-Encoding Instructions

The depth task requires a nonlinear transformation from metric depth to a scalar u, followed by a seven-segment RGB trajectory from black to white without per-image range normalization. The surface-normal task likewise maps each normalized component from $[-1, 1]$ into RGB while imposing the local direction convention $n_z \leq 0$. Only 20.62% of native depth-output pixels lie within an RGB distance of 0.02 from the prescribed color trajectory, whereas 41.29% deviate by more than

0.1. For surface normals, 6.32% of native pixels violate the $n_z \leq 0$ convention, and 5.98% encode vectors whose lengths differ from one by more than 0.1. These results show that GPT-Image-2 does not follow the multi-stage numerical and color-space instructions consistently.

The image-level Pearson and Spearman correlations between color residual and AbsRel are only 0.069 and 0.054, respectively. Moreover, pixels lying within 0.02 of the legal color trajectory still have a pooled AbsRel of 0.7926 and a per-image macro AbsRel of 0.7508, indicating that valid-looking colors can still encode incorrect geometry. Conversely, directly applying the prescribed 8-bit encoding to ground-truth depth yields an AbsRel of 0.001178 and δ_1 of 100%, reproducing the resizing operations used by the prediction pipeline yields an AbsRel of 0.001942 and δ_1 of 99.9995%. Thus, GPT-Image-2 exhibits both rule-following errors and errors in the underlying geometric estimates, neither factor alone explains the observed performance.

D.3 Implications for Generative Foundation Models

These limitations suggest two directions for improving generative foundation models beyond the tasks considered in this work. First, model pretraining should strengthen perception and representation of the 3D world. The substantial gap between GPT-Image-2's competitive performance on 2D pixel prediction and its depth and surface-normal results indicates that strong image-generation priors do not automatically provide metric geometry, reliable near–far relationships, or locally consistent surface orientation. Future pretraining could incorporate geometry-aware objectives and data, such as depth and normal supervision, multi-view consistency, and explicit spatial reasoning, rather than relying on prompting alone to elicit 3D knowledge.

Second, generative foundation models require stronger semantic grounding and more reliable execution of complex instructions. The difficulty is not limited to the numerical color encodings used for depth and surface normals. As shown in Tab. 1, direct GPT-Image-2 prediction is comparable to Nano Banana Pro and Nano Banana 2 on RefCOCO, where the referring expressions typically identify relatively explicit targets: their cIoU scores are 63.0, 63.7, and 63.3, respectively. The gap becomes clearer on RefCOCO+, where prediction requires stronger appearance-based disambiguation (50.2 versus 53.6 and 53.0), and widens further on RefCOCOg, whose longer expressions require more contextual semantic interpretation (55.7 versus 67.1 and 63.9). This trend is consistent with the encoding diagnostics: as instructions require more semantic interpretation or more precise multi-stage execution, GPT-Image-2 becomes less reliable. Therefore, improving generative foundation models for unified pixel prediction requires jointly strengthening 3D-aware pretraining, semantic understanding, and robust instruction following.