VisionAgent: Fine-Grained Image Editing with LLM Reasoning and Classical Computer Vision

Sawyer Rice
UC Santa Barbara
sawyerrice@ucsb.edu

Nik Belle
UC Santa Barbara
nbelle@ucsb.edu

Dakota Barnes
UC Santa Barbara
dakotabarnes@ucsb.edu

***Abstract*—Current AI driven image generation models lack significantly in the domain of image editing. Approaches such as OpenAI's GPT-Image, Google's Nano Banana, and Black Forrest Lab's Flux-Kontext generate complete images from scratch when tasked with modifying regions within an existing image. In our work, we explore an alternative approach that places careful attention on maintaining the integrity of background or non-edit regions, while also maximizing perceptual quality of edits. To accomplish this, we propose VisionAgent, an OpenCode coding agent with access to specialized computer vision tools spanning the domains of simple object transformation to object detection and segmentation. Through specialized agent prompts, we give VisionAgent structured guidance on how to accomplish both simple and multi-step editing tasks. In our work, we also develop a benchmark of 27 tasks to help evaluate VisionAgent's performance against state of the art approaches. To quantify both edit accuracy and background preservation, we identify the two representative metrics, LPIPS and preserved percent changed. Structuring our study around these two metrics, we demonstrate VisionAgent's ability to compromise between both metrics to deliver high quality image edits that are faithful to the source images they derive from.**

I. Introduction

Image editing is a fundamental capability that modern computer vision tools are evaluated on. As instruction-guided editing models improve, fields that previously could not rely on them are starting to reconsider, evaluating them for domain-specific tasks that require greater attention to detail [1]. However, the evaluation lens now shifts to assess visual accuracy as much as visual realism. This is because in practical settings, such as graphic design, the goal is not to generate a new image that just incorporates the suggested change cleanly. It is to apply a scoped change while providing high confidence that the rest of the image is untouched. For example, a user might want to shrink a label or move a figure in a presentation without disturbing the overall layout. Even a small unintended change outside the defined edit region can alter the information being communicated, which is a risk that settings like medical or scientific visualization are likely not willing to take.

Significant progress has been made in using diffusion-based and multi-modal image editing models for instruction-guided editing scenarios [2]–[5]. They can produce visually-passing outputs when given a prompt like "make the text blue." But while their abilities to generate holistic images that blend in changes are considered a strength under the evaluation lens of previous papers, they do not provide high confidence that non-target regions will remain unchanged. In practice, even state of the art models often alter typography, spacing, object boundaries, or background content that a user never asked to modify despite shallowly satisfying the user's request [6]–[8].

For fine-grain edits like recoloring, resizing, or moving objects and text this limitation becomes more of a focus in evaluation. In these situations there is a clearly defined target region, a correct transformation that requires minimal creativity, and a high expectation that everything else should be preserved. Realism alone is not enough of an evaluation metric. Controllability is what truly matters. The user wants to know if the system identified the right edit region, applied the edit accurately, and that it preserved the rest of the image with high confidence.

Interaction-based methods provide more user control by requiring masks or structural guidance as inputs [9], [10], but this adds too much of a burden to be a practical solution. POEM [8] attempts to reduce this burden by using a multi-modal LLM for visual grounding, transformation prediction, and guiding inpainting across a fixed pipeline. This fixed pipeline, though, makes their system unable to use tools in different orders and catch errors throughout the editing process. They also still rely on generative synthesis for the final image, causing lower confidence in background artifacts being preserved. Finally, general purpose coding agents offer a more flexible approach by writing image-editing code on-the-fly, however they lack the fine-grain attention to detail when defining these tools and must infer manipulation procedures, which may be too brittle for complex tasks.

In this work, we argue that precise image editing in scenarios where high confidence in background and edit region preservation are critical is a task best fit for a tool-using agent, rather than an end-to-end generative tool. We present VisionAgent, which combines an LLM-powered general purpose coding agent with classical computer vision tools to execute localized and deterministic image transformations. While the multi-modal LLM handles interpreting the task, selecting the focus region, and planning its execution pipeline, the tools are responsible for carrying out the edits, such as scaling, translating, inpainting, and recoloring. This setup allows for preserving the edit region and background more reliably than a purely generative approach.

We explore this problem using presentation-style images, where users often want to perform targeted changes while

preserving the slide content and layout. To rigorously evaluate our agent's success and compare it to state of the art approaches we introduce a benchmark with 27 hand-crafted image editing tasks of this style, and metrics that separately measure region preservation and edit accuracy. Finally, we compare the performance of end-to-end generative models, a generic coding agent, and our agent that uses computer vision tools to highlight where each approach succeeds and fails on targeted image editing tasks.

A. Contribution

We make the following contributions:

1) **A benchmark for precise image editing** with practical task categories (recolor, resize, move) of varying difficulties and multiple image contexts, ground-truth targets, and metrics that separately evaluate edit accuracy and region preservation.
2) **A specialized image-editing LLM agent** that uses computer vision tools for localized image manipulation.
3) **A comparative study** across end-to-end generative models, a generic coding agent, and our VisionAgent, showing that deterministic tool-use yields substantially better image preservation on precise editing tasks.

II. Related Works

A. Text-Based Instructional Editing

Instruction-guided editing with guidance from image and text was pioneered by InstructPix2Pix [3]. MGIE [11] explored using an MLLM to improve instructions before passing them into a diffusion model. SmartEdit [12] explored handling more complex instructions that require stronger reasoning. TurboEdit [5], LEDITS++ [4], and DiffEdit [6] explore using cross-attention masking, few-step diffusion, and other techniques. This category of work is fundamentally limited in performing precise geometric transformations with background preservation as they rely on diffusion cross-attention for spatial control.

B. MLLM-Guided Editing Pipelines

The closest work to ours is POEM [8] as they use a multi-modal LLM to understand an image, Grounded-SAM [13], [14] for detection, a math LLM to compute affine transformation matrices, and then perform diffusion-based inpainting. They find that the primary source of error on their own benchmark is using an LLM to predict the affine transformation. GenArtist [15] explored a similar approach, using an LLM to decompose tasks into subproblems with a planning tree.

C. Tool-Using Agents

Visual ChatGPT [16] was the first major paper to explore using an LLM to orchestrate VLMs to perform various question-answering tasks. ViperGPT [17] followed up on their work with an LLM that generates Python code to compose vision APIs. Our work extends these works by focusing on targeted image editing, assessing performance with our benchmark designed to measure edit precision and region preservation.

III. Methodology

A. Benchmark

1) Tasks: In our benchmark, we wanted to model a realistic scenario where a user has a finished presentation but needs to make a last-minute targeted change, such as recoloring a label, resizing a figure, or repositioning an object without disturbing the layout. We created 27 unique tasks, where each consists of an input image (960×540 pixels), a ground-truth output manually produced in Google Slides resembling the expected change, and a natural-language prompt outlining one of these targeted changes.

a) Template images: We utilize three different presentation-style template images, with each emphasizing different visual content to diversify the challenges we provide to each system we evaluate (Fig. 1).

- **Image 1 (Text-heavy):** A slide introducing a team that is dominated by a large text heading, colored tabs, and a busy background. The majority of tasks created for this image involve identifying and manipulating textual elements in a busy layout.
- **Image 2 (Shapes):** A slide featuring multiple solid colored geometric shapes (triangles, squares, circles) alongside text rows. The majority of these tasks trend towards applying transformations to the shapes, testing the system's ability to isolate and manipulate geometric objects.
- **Image 3 (Photos):** A slide that contains two realistic photographs of a tower and a mountain, colored tiles, and text. Difficult tasks for this template involve identifying and manipulating these photographs and panels as a whole, requiring the system to understand the complex imagery in a structured layout.

b) Task categories: We define three categories of edits, each testing a distinct type of transformation:

- **Recolor:** Change the color of a specified element (e.g., text, shape, or panel) to a defined RGB value. This tests how well the system can identify the correct region, modify only its color, and preserve the geometry and surrounding content.
- **Resize:** Make an element bigger or smaller, often with an anchor constraint to reduce chances of under-specification. This tests how well a system can apply geometric scaling and reconstruct the background behind the original region.
- **Move:** Translate, rotate, or relocate one or more elements to specific locations. This tests spatial reasoning, coordinate understanding, and background inpainting after displacement.

c) Difficulty levels: For each image and category pair, we define three difficulty levels that progressively require better contextual reasoning and multi-step execution:

- **Easy:** A single transformation on a clearly specified object. For example: *"Make the triangle in the middle of the image red (255,0,0)."* These tasks assess how well

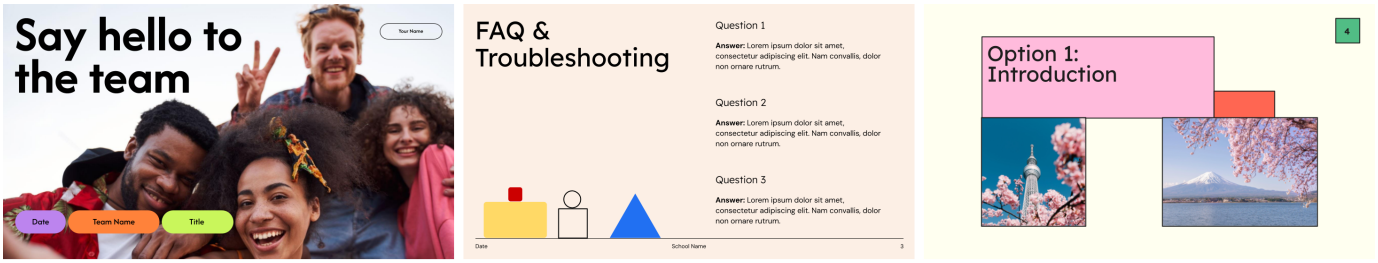

Image 1 (Text-heavy) Image 2 (Shapes) Image 3 (Photos)

Fig. 1. The three template images used in our benchmark (each 960×540 pixels).

the system can identify an object and perform a single editing operation.

- **Medium:** A transformation where either the target object or transformation is defined relative to another element in the image. For example: *"Make the shape that is tallest with respect to the other shapes red (255,0,0)."* The system must compare objects before applying the edit, requiring semantic understanding and spatial reasoning.
- **Hard:** A task that involves multiple objects or sequential steps, that must be applied in a proper order. For example: *"Move the image on the right to be where the image on the left is, and move the image on the left to be where the image on the right is. Make sure they both maintain the same top left corner position."* In this case, if the system is an agent, it must plan the proper execution order and perform background reconstruction at both locations.

2) Evaluation Metrics: The main problem we are focusing on with this work is the fact that modern "automatic" image-editing tools that allow for flexible prompting do not ensure high confidence that objects outside of the edit are preserved. To reflect this, our benchmark evaluates precise image editing across two independent axes: edit accuracy and non-edit region preservation. In other words, whether the system applied the correct transformations and also left everything else untouched. In developing our benchmark, we quickly found that evaluating the whole image with metrics like SSIM or PSNR conflates these axes. For example, a model that perfectly preserves the background but botches the edit would still score well overall on SSIM because the preserved region dominates the pixel count. Therefore we carefully selected the most valuable metric to represent each axis and then combine them into a single composite score. All tracked metrics are reported in Table IV in the Appendix.

a) Edit mask derivation: Any metrics we collect on edit regions rely on a binary mask that separates the edit zone from the preserved zone. We calculate the per-pixel maximum channel difference between the original image and ground truth with a threshold at 20 intensity levels to identify changed pixels. We then dilate this mask by 15 pixels to account for anti-aliasing at the edit region boundaries that would otherwise be considered as part of the preserved region. The preserved mask is the complement of the edit region.

b) Edit Region LPIPS: The first major metric we report is learned perceptual image patch similarity (LPIPS) [18]. We start by cropping the edited image and the ground truth to a bounding box that surrounds the edit mask. This is to confine the evaluation to the edited artifacts. Then we compute LPIPS using a pretrained VGG network, which finds the distance between two images in deep feature space. This metric is meant to show perceptual similarity beyond pixel-level agreement. A score of 0 would indicate that two images are identical. We decided to use LPIPS as our main edit accuracy metric, over SSIM or MAE, as it captures the structural and color fidelity of transformations like recoloring, resizing, and moving where small pixel shifts can inflate pixel-based error while being perceptually negligible.

c) Preserved % Changed: For the second axis, we want to measure background integrity. To do this we compare the edited image to the original at every pixel in the preserved mask. A pixel gets counted if the maximum channel difference exceeds 10 intensity levels and we report the percentage of pixels changed from this region. This metric is meant to reflect exactly what a user cares about most in the problem setting of our benchmark as it represents the fraction of content that was inadvertently modified. We chose to use this as our main preservation metric instead of preserved SSIM because it gives a more interpretable and structured measurement. We also decided not to use IoU as it conflates the two situations where a system edits too small of a region and also the wrong region. In a situation where any unintended change is consequential, we want every preserved-region pixel change to have a high impact on the score. End-to-end models, or systems that regenerate the whole image, score poorly here while approaches that only modify the target region perform much better.

d) Composite score: To offer an overall ranking of each model, we combine both axes into a single score with equal weighting:

$$S = g \cdot \frac{1}{2}\left[(1 - \text{LPIPS}_{\text{edit}}) + \left(1 - \frac{P}{100}\right)\right] \tag{1}$$

where $\text{LPIPS}_{\text{edit}}$ is the edit region LPIPS, P is the preserved % changed, and g is a binary gate that is 0 if fewer than 0.1% of the total pixels changed (indicating that no edit was attempted) and 1 otherwise. This gate ensures that a run where no change takes place receives a zero instead of being rewarded for preservation. Both terms are bounded in $[0, 1]$ with higher values indicating better performance. We decided

to weight edit accuracy and preservation the same because both requirements are essential for a successful edit. A system under test must apply the correct transformation and leave the rest of the image unchanged. This composite score is created directly from the edit region LPIPS and the preserved percent change reported individually to make evaluation easy to interpret.

e) Additional metrics: While those are the three major metrics reported in our benchmark, we also compute several supplementary metrics for further diagnostic, which can be found in Table IV in the Appendix. Edit region MAE gets the mean absolute pixel error between ground-truth and edited images within the edit region, which is particularly helpful in evaluating color accuracy when a recoloring task has a specified RGB value. Both overall LPIPS and SSIM showcase image-level comparisons of the edited versus the ground-truth, capturing global quality at the cost of conflating the two axes. Looking at preserved region SSIM and MAE offer different views that are sensitive to structural degradation and color differences respectively. We chose to omit these metrics from the main comparison table to focus on what we deemed the most valuable metrics to represent the two independent axes of edit accuracy and region preservation and make the results easy to understand.

3) Model/Agent Comparison: We compared 5 different solutions for precise image editing to evaluate our solution. We first test generative models, which are called directly from the provider's API. We choose GPT-Image-1 from OpenAI, Gemini-2.5-Flash-Image from Google (also referenced as Nano Banana), and Flux-Kontext-Pro from Black Forrest Labs [19]. GPT-Image-1 and Gemini-2.5-Flash-Image are auto regressive multi modal models with image generation capabilities, while Flux-Kontext-Pro is diffusion based. Additionally, we choose to test the default OpenCode agent with standard python libraries as an agentic baseline for VisionAgent. Lastly, we attempted to evaluate POEM [8], the most closely related work. However, POEM's pipeline failed to produce any valid edits on a subset of our tasks, likely due to our presentation-style task images with dense text and geometric shapes rather than natural images with distinct foreground objects.

4) Harness: To facilitate the testing of our VisionAgent and existing solutions, we created a benchmarking harness to execute any agent or model on our tasks. The harness CLI (command-line interface) can be configured to accept task filters, agent filters, and running each configuration n times across multiple threads. The harness will 1) create a log directory for each data point with the task image and prompt, 2) call the agent's `execute.py` entry point to generate the image, and 3) compute the metrics of the changed image to be saved in `result.json`. This flow enables each data point to be contained, reproducible, and easily runnable for quick development.

B. VisionAgent

1) Coding Agent Backend: VisionAgent is built on OpenCode, an open source command-line coding agent. It is favored by developers as an open source alternative to coding agents such as Claude Code or Codex. OpenCode is built to be LLM agnostic, and has access to a variety of tools such as bash, edit, write, read, etc. OpenCode functions by looping the LLM queries with the tool call's output for continuous iteration (the ReAct framework [20]). This can be advantageous compared to a fixed pipeline like POEM [8], as OpenCode is able to inspect results, catch errors, and iteratively refine based on environment feedback.

We choose Claude Haiku 4.5 as the model to power VisionAgent for two reasons. The first is due to its balance of low cost and high agentic capabilities. To truly test capabilities of our agent, we could perform more extensive testing with different higher-cost models such as Claude Opus 4.6, or other models from OpenAI or Google. The second benefit to Claude Haiku 4.5 is its multi-modal capabilities, allowing the agent to directly view the input image and intermediate results. This enables the agent to not just make changes to the image with tools, but also "see" and validate the changes.

2) Custom Tools: To operate on images, VisionAgent uses a set of specialized tools spanning the domains of detection, segmentation, recoloring, translation, rotation, and infilling.

- `detect_text`: Using EasyOCR, this tool identifies all regions within the image containing text. A helper is also used to discern text color as well as background color for all detected text regions. For each input image, the tool returns a list of the four corners of each text filled region, the color of the text in the region, and the background color of the text filled region.
- `detect_object`: Detect objects using Grounding DINO zero shot detection model. Relevant object categories or labels are collected from the user prompt and are passed into the detection model. Bounding box locations, labels, and confidence scores are returned back to the agent for later use in the pipeline.
- `create_text_mask`: This tool utilizes the background and text colors to create a grayscale map outlining all relevant text in a region.
- `segment_object`: Bounding box coordinates along with the original image are passed in to Meta's SAM model to produce several candidate object masks for the region. The candidate mask with the highest IoU is chosen and saved to a file location echoed back to the agent.
- `fill_noise`: Given an object or text mask, the tool replaces all regions of interest with noise. The noise is not purely random, however. After taking several samples from the background of the mask, noise is generated using a gaussian distribution centered on the mean RGB value of the surrounding pixels.
- `diffuse_inpaint`: After filling a region with noise, this tool is called upon to utilize OpenCV's Navier-Stokes diffusion method to smooth the noisy region into the background. The diffusion is specifically localized around an input region of interest to ensure the rest of the image remains unchanged.

- `translate_object`: Based on an input bounding box or mask (if available), and input transformation distances in the x and y directions, an input object is translated within the image. A simple blending is applied to the edges of the object to give the impression of a seamless modification.
- `rotate_object`: Similar to `translate_object`, but with an angle of rotation parameter rather than an x and y translation.
- `recolor_pixels`: This tool allows for both smooth color blending in objects with details and edges, as well as direct replacement with an input RGB value. An input mask is provided to closely guide coloring.
- `resize_region_simple`: Using an input scale factor determined by the agent, a region in the original image is scaled either up or down. Bounding box coordinates, object destination coordinates, and an optional mask are also provided in addition to the scale factor.
- `inspect_region`: This tool assesses dominant colors and brightness within a bounding box defined zone.
- `save_image`: Following a string of transformations, `save_image` is called to store the final version of the image.

TABLE I
CUSTOM TOOLS AVAILABLE TO THE TOOL-AUGMENTED AGENT.

Tool	Function
`detect_text`	Locate text regions (bboxes, colors)
`detect_object`	Locate object regions (bboxes, labels)
`create_text_mask`	Isolate text pixels via thresholding
`segment_object`	Isolate object pixels via SAM
`fill_noise`	Noise infilling prior to inpainting
`diffuse_inpaint`	Background reconstruction
`translate_object`	Move object in x, y directions
`rotate_object`	Rotate by angle
`recolor_pixels`	Apply new color to masked pixels
`resize_region_simple`	Scale bounded region
`inspect_region`	Analyze region properties
`save_image`	Output the result

3) Custom Prompts: The custom agent prompt is the driving force orchestrating VisionAgent's interactions with the defined tools. Primarily, the agent prompt outlines the general plan the agent should follow when working towards a goal.

First, the agent is instructed to assess the user prompt to determine if text, objects, or both will need to be detected to accomplish the goal. If objects are required, the agent is instructed to extract all relevant categories and keywords from the user prompt to pass to the object detection tool.

Upon detection of all relevant features in the image, the prompt instructs the agent to proceed with all necessary mask generations that will be later required for translations, scalings, recolorings, and rotations.

Once all important features have been extracted and masked properly, the original image can start to be prepared for transformations. The agent prompt now guides VisionAgent to the `fill_noise` and `diffuse_inpaint` tools. The prompt specifies that the tools be called in this order as the diffusion inpainting is largely dependent on the input region being qualitatively similar to the background around it. After iterating over all masks and repairing the original image, the image is finally ready for transformations.

The agent prompt now encourages the agent to proceed with all translations, rotations, scalings, and recolorings before submitting the final image to the save image tool.

In the agent prompt, specific instructions are given for handling complex multi-step requests. Specifically, the agent prompt instructs the agent to complete all detections and mask generations in parallel. Then, all image cleaning and repairs are to be done exclusively in sequence to prevent repairs from different parts of the user prompt interfering with others. Finally, transformations are also required to be chained together sequentially to ensure operations are not lost or mangled in intermediate representations.

IV. RESULTS

We evaluate across six agents for a comprehensive comparison. As shown in the rows of Table-II, we have the Baseline (original task image without edits), GPT-Image-1 and Gemini-2.5-Flash (auto-regressive multi-modal), Flux-Kontext-Pro (diffusion based), OpenCode Agent with claude-haiku-4.5 (agent baseline), and finally VisionAgent with claude-haiku-4.5 (ours). The columns include the LPIPS score (Learned Perceptual Image Patch Similarity), the Pres % Δ (Preserved Percent Changed), and the Composite score (the final grade). The results in Table-II are averaged across all 27 tasks, and each agent is evaluated only once per task (n=1) to conserve cost. Additionally, Table-III shows how each agent's composite score compares across the different task types of Move, Recolor, and Resize.

TABLE II
OVERALL AGENT PERFORMANCE AVERAGED ACROSS ALL TASKS.

Agent	LPIPS ↓	Pres % Δ ↓	Composite ↑	Time (s)
Baseline (no-edit)	0.381	0.000	0.000	.1
GPT-Image-1	0.498	50.315	0.500	30.4
Flux-Kontext-Pro	0.435	22.725	0.669	12.7
Gemini-2.5-Flash	0.348	18.613	0.733	**9.5**
OpenCode Agent	0.190	4.331	0.820	14.1
VisionAgent (ours)	**0.186**	**0.457**	**0.905**	36.5

A. Baseline Performance

At a first glance, we can see that VisionAgent (ours) outperforms the existing solutions on our metrics, reaching the highest Composite score of 0.905, 10.3% above the closest competitor, the OpenCode Agent.

GPT-Image-1 saw significant struggle with edit accuracy because the model failed to maintain the original dimensions of the input image (940x540). GPT-Image-1 changes the resolution of produced images to 1536x1024 or 1024x1024 mangling the structural integrity of the original image. Further, GPT-Image-1's reliance on a complete regeneration of the image is clear as the model appears to take creative liberties

with features all around the image, not just in reference to the edit regions, reflected in the preserved percent changed score of 50.315 in Table-II. As shown in Figure-3 for the "Recolor Hard Img1" task, GPT-Image-1 is able to reason about the second and third word in the sentence "Say hello to the team" and change them to their prompted colors of blue and red respectively.

Flux-Kontext-Pro had slightly different issues, such as returning an unedited image, going astray and making unrelated edits, and also resizing the image to 1392×752. This fact is expressed in Flux's LPIPS score of .435, second only to GPT-Image-1 for the worst accuracy of the assessed methods. In some cases, Flux-Kontext-Pro took the initiative to completely regenerate individual aspects of the images before reconstructing the final image with a set of disjointed and incoherent parts. For example as we can see in Figure-4 for the "Recolor Hard Img2" task, Flux-Kontext-Pro turns the blue triangle into a long rectangle instead of recoloring the stacked boxes as the prompt specifies.

Gemini-2.5-Flash saw a significant improvement in both LPIPS score and qualitative accuracy compared to the previous two approaches, especially at identifying objects and regions relevant to the prompted edits. Similarly to the other models, Gemini-2.5-Flash rescales the original image to 1344x768. However, Google's model proved weak when tasked with scaling and translations, especially when these transformations were relative to other features in the image. Figure-5 highlights that for all of the "Move ... Img3" tasks, Gemini-2.5-Flash (nano-banana) does not attempt to move any of task images according to the task prompt. This seems to be a result of Gemini-2.5-Flash being unable to reason about an image frame that is inside of an image (slide shows).

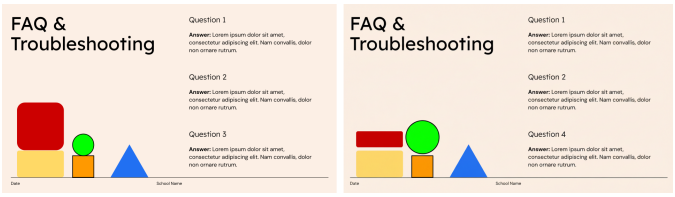

Fig. 2. Resize Hard Img2: Ground Truth (left) vs. Gemini-2.5-Flash(right). Prompt: "Make the red square as wide as the yellow rectangle. Align the bottom edge of the new red square with the top edge of the yellow rectangle. Move the new enlarged red square so that the bottom edge of the red square is touching the top edge of the yellow rectangle. Also, make the green circle as wide as the orange square. Align the bottom edge of the new green circle with the top edge of the orange square so that it looks like the green circle is on top of the orange square." Gemini-2.5-Flash struggles to accurately edit objects when the transformations require careful inspection of other objects in the image.

The OpenCode Agent saw a massive improvement from the LPIPS score of Gemini-2.5-Flash. Now able to create its own specialized tools per task, the agent can maintain aspects of the original image with targeted edits, and leaving the image in the original size (940x540). However, with such freedom, the OpenCode Agent was vulnerable to exploring poor approaches in some cases. Rewriting all of its code on each agent run, the OpenCode Agent was extremely inconsistent as successes from previous trials were not reflected in later stages. This is clear in Figure-3 on the "Recolor Easy Img1" task, as you can see OpenCode Agent creates a bounding box over the text and the people in the image turning them blue, instead of just the text as the instructions specified. However, OpenCode did have a low LPIPS score of 0.190 as shown in Table-II, primarily due to the agents ability to make edits to the image without regenerating all pixels from scratch.

TABLE III
COMPOSITE SCORE BY TASK TYPE.

Agent	Move	Recolor	Resize	Overall
Baseline (no-edit)	0.000	0.000	0.000	0.000
GPT-Image-1	0.543	0.506	0.449	0.500
Flux-Kontext-Pro	0.671	0.709	0.626	0.669
Gemini-2.5-Flash	0.675	0.821	0.703	0.733
OpenCode Agent	0.735	0.836	**0.887**	0.820
VisionAgent (ours)	**0.882**	**0.980**	0.852	**0.905**

B. *VisionAgent Performance*

When specifically analyzing how VisionAgent performed on the benchmark, we can start by looking at its quantitative results in Table-II. As expected, VisionAgent achieves a significantly higher composite score (0.905) and per-category LPIPS score (0.186) than generative models for two main reasons. First, VisionAgent has the ability to interact with the image across multiple steps to ground itself in the image's spatial context. It can also self correct throughout the process by double checking its work. We can ground this in the quantitative results, with a good example captured in Fig. 4 on the "Move Medium Img2" task. Through multi-step interaction with the image, VisionAgent is able to identify where the "t" in "Troubleshooting" is located, segment the triangle, move it to the proper location, and blend the background to ensure no artifacts of the triangle are left over. The OpenCode agent also performs this tasks better than the generative models, however it doesn't have a predefined text detector which forces it to rely on more shallow object detection tools that it writes for itself. Interestingly, the OpenCode agent scores higher than the VisionAgent for resize tasks. This is likely due to poor blending from the VisionAgent causing a high LPIPS score.

The metrics where we see the strongest performance of VisionAgent across the board is the preserved region percent changed, with an overall score of 0.457. It has the natural advantage over generative models since it leaves the preserved region almost fully untouched when the MLLM correctly identifies the edit region, while the others will always slightly modify preserved pixels since they regenerate the full image. We see stronger performance than the basic OpenCode agent as well, likely due to a blend of explicit prompting and tools that allow it to have more fine-grain control over the image. In the "Resize Medium Img3" example of Fig. 5, we can see VisionAgent actually failing within the edit region by forgetting to recolor the left panel red. However, it leaves the preserved region completely unchanged while OpenCode

hallucinates and recolors the text above to be red. Examples like this highlight that even with the ability to perform targeted changes across steps, agents need to be steered in the right direction with proper tools and prompting to ensure that they do not hallucinate changes to regions that are meant to be preserved.

V. Discussion

A. Strengths

VisionAgent excels at targeting specific subsets of an image when making modifications. Alternative editing approaches may feed the entire image into a generative model, thus gambling with the integrity of regions of the image completely unrelated to the edits. VisionAgent instead uses traditional CV tools to identify and extract regions of interest before making edits. As a result, VisionAgent sees favorable results with respect to metrics measuring overall likeness with the original image. Through the use of specialized tools like `detect_text`, `detect_object`, and `segment_object`, VisionAgent is able to gather globalized image context regarding the orientation of key features in the image. Using this enhanced world-view, VisionAgent can more carefully target modifications than any competing diffusion-based approaches.

Further, VisionAgent's iterative nature lends itself more flexibility in terms of error recovery similar to MIRA [21]. While not implemented as a specific tool or function, error recovery is inherently achieved through allowing the agent to continuously iterate until it believes it has sufficiently modified the image. While iterating, VisionAgent is capable of identifying and addressing unintentional shortcomings and deviations from previous steps in an image's life-cycle.

Also owing to VisionAgent's iterative approach, VisionAgent sees great success in applying complex multi-step prompts to images. Generating intermediate representations between atomic actions, VisionAgent is able to methodically navigate through different stages of the editing process without losing sight of the ultimate goal.

Next, a major advantage of using an agent for performing targeted image edits is explainability. It is difficult to understand why end-to-end generative models fail, and how we can improve them in specific scenarios. With an agent, we can see directly into its decision making process to understand where in particular it is going astray. For example, when VisionAgent recolors text it struggles to pick out specific words when the whole sentence is grouped closely together. We were able to understand this shortcoming by looking through its logs and thus we knew where to start when looking to improve its performance. On the other hand, we have no idea why a model like Nano Banana changes text to the wrong shade or modifies more text than the prompt states.

Finally, we believe our benchmark offers new insights into state of the art image generation tools by testing them in a more practical image editing setting. While they are often spoken highly of for their improving generative abilities, they are not often stress tested for how well they preserve regions of the image that are not intended to be changed. Our benchmark showcases significant shortcomings in not only the generative models, but also agents; thus shining light on areas with clear room for improvement.

B. Limitations

When exposed to scenarios that require complex semantic understanding, VisionAgent often falls short of other diffusion based approaches. Currently, VisionAgent uses a naive diffusion method for infilling localized regions around the image, specifically in cases where objects are moved around the image. When tasked with repairing object imprints over a complex background, VisionAgent's infilling tools are not sufficient to infer nuanced patterns or coloring schemes. This is likely what leads to the standard OpenCode agent actually performing better than VisionAgent on the resize tasks. For a case such as shrinking a word on a solid color background. VisionAgent's method of carefully segmenting the word and going through the steps to infill the background sometimes causes smear that would link to a lower region LPIPS score than a naive approach from OpenCode where it simply cuts a box around the word and shrinks it. This is a flaw in both the benchmarking setup and our agent.

Another limitation we noticed throughout the process of working with an LLM agent was the typical whac-a-mole problem. When defining a powerful, multi-step tool an agent might either have its performance enhanced immensely or completely fail due to not having enough freedom for the LLM to decide what tools to call. Alternatively, having multiple simple tools gives an agent the ability to use them in whatever order it wants. However, if the LLM doesn't come up with a solid plan it might get overwhelmed by the high number of tools and fail. It gets more difficult when tools can very easily be overfit to specific tasks, degrading performance on previously passing tasks. We ran into this situation often, noticing that passing tasks would regress after making small changes to the agent's prompt or a tool. While using an LLM-agent does offer high confidence in background preservation, agents are constrained by how well their prompts and tools balance both generalization and specificity.

Next, the runtime of our agent was longer than any of the other baselines. This is an inherent limitation of agents as they execute in a reason + action loop until the LLM deems itself finished.

Looking at the benchmark, instead of aggregating many metrics we chose to incorporate just two in our composite score. While we value the explainability and even split between the single most important score for each of the two axes, a composite score that is dependent on more metrics would offer more attention to different specific qualities of the image. For example, specifically focusing on MAE of the edit region would give us a tunable focus on raw pixel intensity. However, bringing more metrics into the composite score would create the new subjective problem of how much to weight each of them. Additionally, by incorporating more

terms in the equation, the score becomes less explainable, so we see it as a tradeoff.

C. Future Work

Our work leads to many potential future directions, particularly in improving our agent and benchmarking.

While VisionAgent showed clear improvement over generative models and a basic coding agent in many examples, there are still many failing scenarios. We would like to explore using more advanced inpainting and segmenting techniques to improve in scenarios where the masks do not capture the full intended edit region. Additionally, giving the agent more tools and better prompts for spotting intermediate errors could yield better success through self-correction. If we had access to more expensive models, like Claude Opus 4.5, it would be interesting to evaluate if more or less structure around the coding agent leads to more success. There is a chance that too many tools would hinder performance when using a more powerful LLM that can discover and implement better tools on its own. Finally, a clear spot where our agent fails is in segmenting specific parts of a long piece of text. Using smarter text detection would allow for more find-grain edits of words in a longer sentence.

For the benchmark, we attempted to capture different image styles, multiple types of image edits, and varying difficulties of prompts that test the system's ability to understand spatial context and apply multi-step transformations. Including more images with unique prompts would be the most clear way to improve the diversity of the benchmark. It currently identifies major shortcomings of state-of-the-art tools, but stronger insights could be extracted with more diverse task categories beyond the three we defined. Finally, while we aim to offer high explainability with our metrics by providing a holistic score with only two terms, it could be valuable to provide a second composite metric that incorporates MAE and other metrics in a way that properly weights their importance in the evaluation.

Overall, there are many directions we could continue to take this work and we see it as an initial exploration of where current image generation tools fail in precise editing scenarios.

VI. Conclusion

In our work, we present a benchmark that targets the realistic scenario of performing precise image edits using natural language prompts to highlight how accurately different AI systems perform a change within the edit region while also applying minimal unintended changes to other parts of the image. We showcase the shortcomings of state of the art end-to-end generative models on the situations outlined in our benchmark and present this as a task best fit for an LLM agent with classical computer vision tools. We present VisionAgent as an agentic approach that outperforms generative models and a general purpose coding agent on our benchmark emphasizing the value a domain-specific agent provides to this use case. The most important learning we take away from this project is that an agent offers a higher ceiling than end-to-end generative models in precise image transformations while also offer more explainability and self-correction mechanisms. The difficulty in creating a reliable solution comes from the tradeoffs between agent autonomy with many simple tools and a well-defined pipeline with a few powerful tools.

VII. Author Contributions

A. Sawyer

Sawyer primarily worked on improving the tool set and agent prompt across different versions of the agent. He worked on ensuring that the final version of the tool set included sufficiently powerful but general tools focused on singular operations. Most of his work went into stress testing the agent across the tasks and improving the agent and tools without overfitting it to the tasks at hand. He also worked with Nik to create the final task set.

B. Nik

Nik defined the metrics used to evaluate the various agents/models performing the benchmark and implemented the methods used to extract them. He also worked on defining and building the final task set, expanding on the initial ones to incorporate more images and task difficulties. He worked with Sawyer on improving generalization of the agent as the task set expanded and was responsible for getting VisionAgent's final results.

C. Dakota

Dakota created and managed the benchmark harness to evaluate existing state of the art solutions, and to support the quick iteration of tasks, metrics, and agents. He also created the initial benchmark tasks, and the first Vision Agent which were handed off. Finally, he was responsible for collecting and aggregating the final agent and benchmark results for visualization and analysis.

D. Claude Code (AI Usage Disclosure)

We used Claude Code to help us discover related works to our project, quickly iterate on tools and prompts given our ideas, and create visualizations to understand performance. We found it very useful to vet ideas for quick ablations when we clearly outlined the techniques we want to employ. It allowed us to devote our man-power towards more difficult tasks like defining the right computer vision pipelines with the appropriate amount of control given to the agent, investigating the most representative metrics for our benchmark, and crafting high quality benchmark tasks.

References

[1] Y. Lai, W. Qian, B. Liu, H. Li, H. Luo, F. Wang, B. Zhuang, and S. Hong, "Miedb-100k: A comprehensive dataset for medical image editing," 2026. [Online]. Available: https://arxiv.org/abs/2602.09587

[2] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer, "High-resolution image synthesis with latent diffusion models," 2022. [Online]. Available: https://arxiv.org/abs/2112.10752

[3] T. Brooks, A. Holynski, and A. A. Efros, "Instructpix2pix: Learning to follow image editing instructions," 2023. [Online]. Available: https://arxiv.org/abs/2211.09800

[4] M. Brack, F. Friedrich, K. Kornmeier, L. Tsaban, P. Schramowski, K. Kersting, and A. Passos, "Ledits++: Limitless image editing using text-to-image models," 2024. [Online]. Available: https://arxiv.org/abs/2311.16711

[5] G. Deutch, R. Gal, D. Garibi, O. Patashnik, and D. Cohen-Or, "Turboedit: Text-based image editing using few-step diffusion models," 2024. [Online]. Available: https://arxiv.org/abs/2408.00735

[6] G. Couairon, J. Verbeek, H. Schwenk, and M. Cord, "Diffedit: Diffusion-based semantic image editing with mask guidance," 2022. [Online]. Available: https://arxiv.org/abs/2210.11427

[7] B. Kawar, S. Zada, O. Lang, O. Tov, H. Chang, T. Dekel, I. Mosseri, and M. Irani, "Imagic: Text-based real image editing with diffusion models," 2023. [Online]. Available: https://arxiv.org/abs/2210.09276

[8] M. Schouten, M. O. Kaya, S. Belongie, and D. P. Papadopoulos, "Poem: Precise object-level editing via mllm control," 2025. [Online]. Available: https://arxiv.org/abs/2504.08111

[9] Y. Li, Y. Bian, X. Ju, Z. Zhang, J. Zhuang, Y. Shan, Y. Zou, and Q. Xu, "Brushedit: All-in-one image inpainting and editing," 2025. [Online]. Available: https://arxiv.org/abs/2412.10316

[10] L. Zhang, A. Rao, and M. Agrawala, "Adding conditional control to text-to-image diffusion models," 2023. [Online]. Available: https://arxiv.org/abs/2302.05543

[11] T.-J. Fu, W. Hu, X. Du, W. Y. Wang, Y. Yang, and Z. Gan, "Guiding instruction-based image editing via multimodal large language models," 2024. [Online]. Available: https://arxiv.org/abs/2309.17102

[12] Y. Huang, L. Xie, X. Wang, Z. Yuan, X. Cun, Y. Ge, J. Zhou, C. Dong, R. Huang, R. Zhang, and Y. Shan, "Smartedit: Exploring complex instruction-based image editing with multimodal large language models," 2023. [Online]. Available: https://arxiv.org/abs/2312.06739

[13] A. Kirillov, E. Mintun, N. Ravi, H. Mao, C. Rolland, L. Gustafson, T. Xiao, S. Whitehead, A. C. Berg, W.-Y. Lo, P. Dollár, and R. Girshick, "Segment anything," 2023. [Online]. Available: https://arxiv.org/abs/2304.02643

[14] N. Ravi, V. Gabeur, Y.-T. Hu, R. Hu, C. Ryali, T. Ma, H. Khedr, R. Rädle, C. Rolland, L. Gustafson, E. Mintun, J. Pan, K. V. Alwala, N. Carion, C.-Y. Wu, R. Girshick, P. Dollár, and C. Feichtenhofer, "Sam 2: Segment anything in images and videos," 2024. [Online]. Available: https://arxiv.org/abs/2408.00714

[15] Z. Wang, A. Li, Z. Li, and X. Liu, "Genartist: Multimodal llm as an agent for unified image generation and editing," 2024. [Online]. Available: https://arxiv.org/abs/2407.05600

[16] C. Wu, S. Yin, W. Qi, X. Wang, Z. Tang, and N. Duan, "Visual chatgpt: Talking, drawing and editing with visual foundation models," 2023. [Online]. Available: https://arxiv.org/abs/2303.04671

[17] D. Surís, S. Menon, and C. Vondrick, "Vipergpt: Visual inference via python execution for reasoning," 2023. [Online]. Available: https://arxiv.org/abs/2303.08128

[18] R. Zhang, P. Isola, A. A. Efros, E. Shechtman, and O. Wang, "The unreasonable effectiveness of deep features as a perceptual metric," 2018. [Online]. Available: https://arxiv.org/abs/1801.03924

[19] B. F. Labs, S. Batifol, A. Blattmann, F. Boesel, S. Consul, C. Diagne, T. Dockhorn, J. English, Z. English, P. Esser, S. Kulal, K. Lacey, Y. Levi, C. Li, D. Lorenz, J. Müller, D. Podell, R. Rombach, H. Saini, A. Sauer, and L. Smith, "Flux.1 kontext: Flow matching for in-context image generation and editing in latent space," 2025.

[20] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao, "React: Synergizing reasoning and acting in language models," 2023. [Online]. Available: https://arxiv.org/abs/2210.03629

[21] Z. Zeng, H. Hua, and J. Luo, "Mira: Multimodal iterative reasoning agent for image editing," 2025. [Online]. Available: https://arxiv.org/abs/2511.21087

VIII. Appendix

TABLE IV

COMPLETE METRICS FOR ALL AGENTS, AVERAGED ACROSS ALL 27 TASKS. ARROWS INDICATE WHETHER LOWER (↓) OR HIGHER (↑) IS BETTER. BEST VALUES PER METRIC ARE **BOLDED**.

Metric	**GPT-Image-1**	**Flux-Kontext**	**Gemini-2.5-Flash**	**OpenCode Agent**	**VisionAgent**
Edit Region					
Edit LPIPS ↓	0.498	0.435	0.348	0.190	**0.186**
Edit MAE ↓	59.95	67.23	45.01	26.43	**26.12**
Edit SSIM ↑	0.532	0.545	0.623	**0.785**	0.769
IoU ↑	0.099	0.101	0.138	0.516	**0.663**
Preserved Region					
Pres. % Changed ↓	50.315	22.725	18.613	4.331	**0.457**
Pres. MAE ↓	37.629	14.624	11.283	1.897	**0.430**
Pres. SSIM ↑	0.582	0.758	0.787	0.935	**0.951**
Overall					
Overall LPIPS ↓	0.442	0.198	0.187	0.048	**0.034**
Overall SSIM ↑	0.584	0.739	0.768	0.953	**0.961**
Overall PSNR ↑	10.70	14.10	15.99	24.95	**26.79**
Summary					
Composite ↑	0.500	0.669	0.733	0.820	**0.905**
Time (s)	30.4	12.7	**9.5**	14.1	36.5
Cost ($)	0.047	0.040	**0.003**	0.030	0.057

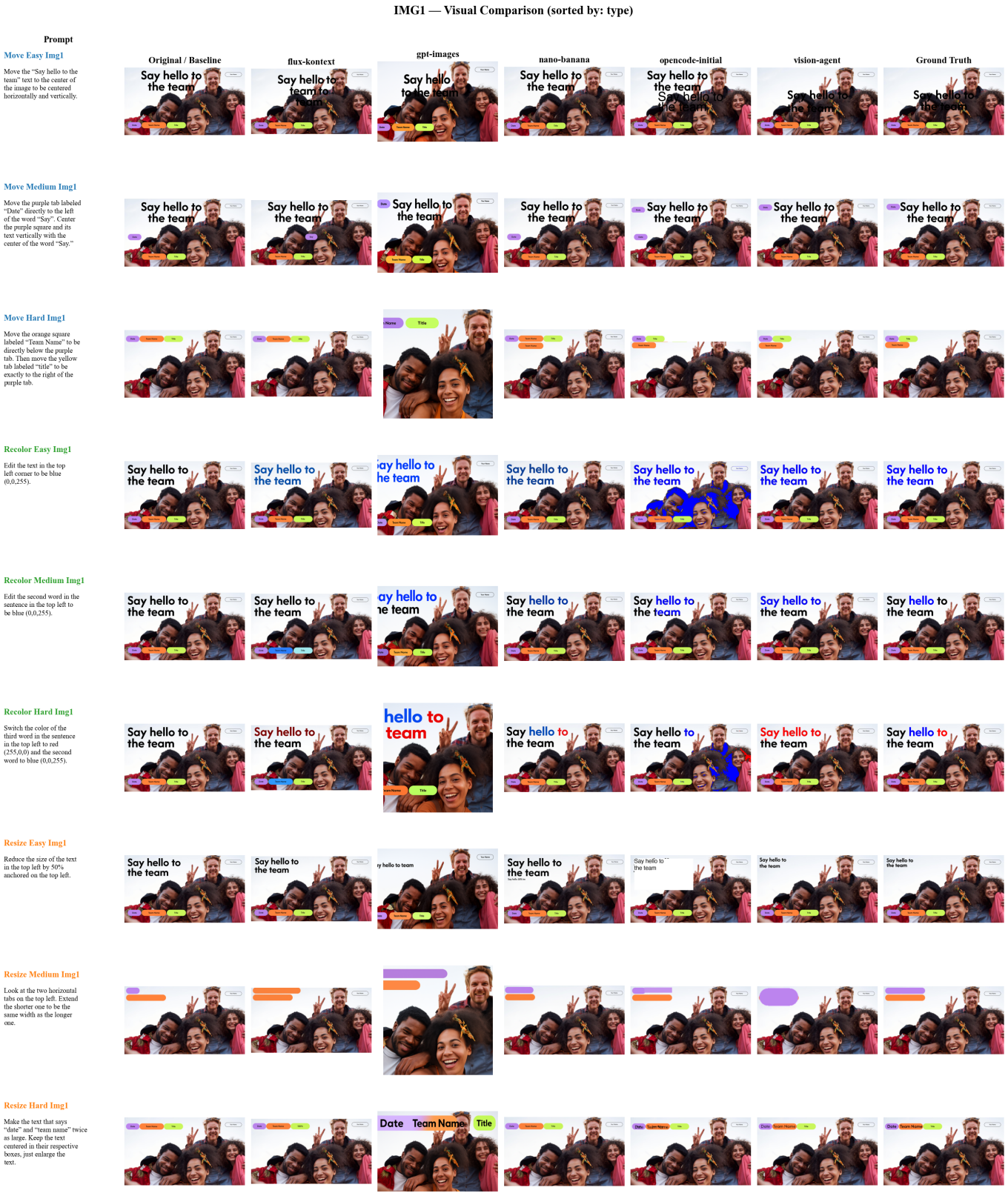

Fig. 3. Img1 Visual Comparison

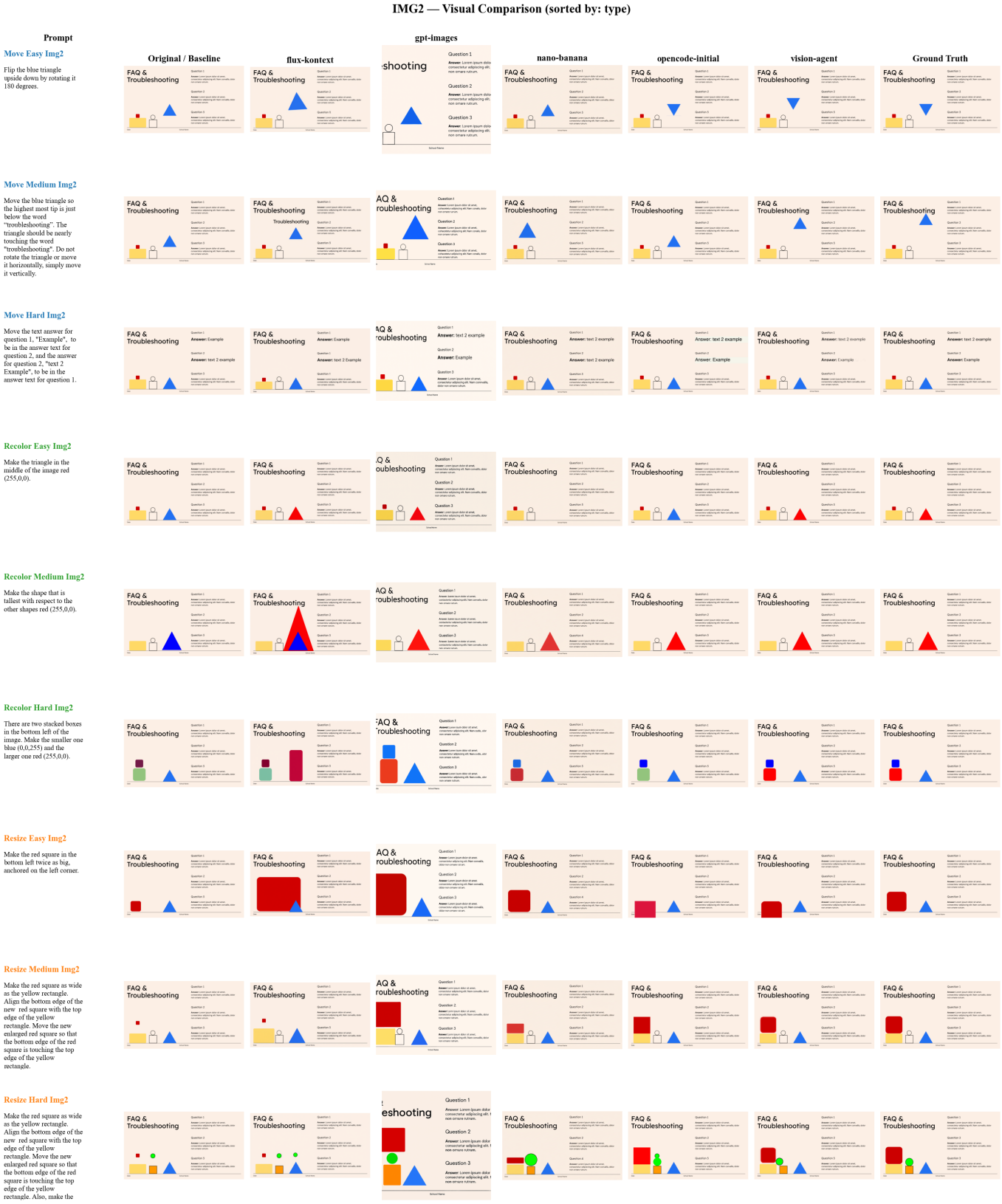

Fig. 4. Img2 Visual Comparison

Fig. 5. Img3 Visual Comparison