Mask-Guided, Training-Free Spatial Control for InstructPix2Pix

Abraham Yirga
Missouri University of Science and Technology

Instructor: Dr. Ce Zhou
CS 5404: Introduction to Computer Vision

November 2025

Abstract

Natural-language diffusion editors like InstructPix2Pix deliver high-fidelity edits but often lack spatial precision, causing unintended background alterations. Training a new mask-aware model is intractable within the project timeline, so this work implements a training-free *Mask-Blended Inference* procedure that couples Segment Anything masks with the pretrained pipeline to enforce edits only inside a user-specified region.

1 Introduction and Motivation

Text-conditioned image editing is now powerful enough to support real creative tasks, yet fine-grained spatial control remains challenging when users ask for localized changes (e.g., 'make the jacket red'). InstructPix2Pix optimizes a single diffusion model that interprets instructions, but it still may bleed onto the background because it never explicitly receives a mask. My proposal targeted this shortcoming by conditioning on a mask, but rather than retraining the diffusion model I realized a runtime blending strategy can fulfill the same promise with a fraction of the time and compute. This report documents the *Mask-Blended Inference* pipeline, explains how it complies with the assignment requirements, and surfaces the reproducible artifacts (proposal, report, code, sample results) that satisfy the CVPR deliverable checklist.

2 Related Work

InstructPix2Pix [1] synthesizes instruction-edit datasets via GPT-3 and Stable Diffusion, training a conditional diffusion network capable of following natural-language directions. Segment Anything (SAM) [2] provides promptable segmentation that isolates objects with a click or bounding box; it ships pretrained ViT backbones and can generate high-quality masks that integrate cleanly with downstream editors. Other mask-aware diffusion papers add mask channels to the U-Net or inject mask-conditioned attention, which requires retraining and GPU resources. The mask-blended inference approach keeps the pretrained weights untouched and enforces spatial constraints at inference time, making it more practical for rapid prototyping or homework settings.

3 Methodology

3.1 Overall Flow

We orchestrate the pretrained behemoths without modifying their weights. SAM produces a binary mask for a user-selected point, InstructPix2Pix generates a baseline edit from the same image and instruction, and a blending step enforces the mask by replacing pixels outside the region of interest with their original values. This flow is deterministic, repeatable, and avoids the GPU-heavy training loops associated with mask-conditioned diffusion.

3.2 Mask-Blended Inference

Let x_0 be the original image and y_T the decoded diffusion output. Given a soft mask $M \in [0,1]^{H\times W}$, resampled to the pipeline resolution, we compute the blended output as

$$\tilde{y}_0 = M \odot y_T + (1-M) \odot x_0.$$

Here M acts like a spatial gate—the only pixels that can change during editing are those inside the mask, while the background inherits the unchanged pixels from x_0. This constraint is applied after the diffusion run, so it does not interfere with sampling but still guarantees that the final output respects the selected region.

3.3 Implementation Notes

The script is located at `submission/code/project_run.py`. It loads InstructPix2Pix from `timbrooks/instruct-pix2pix`, SAM from `models/sam_vit_h_4b8939.pth`, resizes all inputs to 512×512, converts the mask to three channels, and writes the outputs into `results/`. Because the blending happens outside of the diffusion loop, it remains stable even if the base model wanders beyond the mask.

3.4 Pipeline Algorithm

1. Initialize SAM and InstructPix2Pix (loading checkpoints from the `models/` directory).
2. Query SAM with a click near the object to obtain the mask.
3. Run InstructPix2Pix for 20 denoising steps with the user instruction.

4. Blend the denoised image with the original pixels using the SAM mask.

5. Save the original, mask, baseline edit, and blended edit for documentation.

3.5 SAM Mask Preprocessing

SAM expects RGB images at their native resolution, but the diffusion pipeline works at 512×512. We therefore load the original image, query SAM with the clicked point in the original coordinate space, and resize the resulting binary mask to 512×512 using nearest-neighbor interpolation so that sharp boundaries are preserved. The mask is then expanded to three channels and optionally blurred if soft edges are desired, although the current implementation keeps the mask binary to enforce a hard constraint.

3.6 Latent Blending Considerations

The blending could happen inside the latent space, but to keep the implementation simple we blend at the pixel level after decoding. However, the mask is also converted into a 64×64 latent resolution so that future extensions can explore injecting the mask into attention maps or latent noise schedules. The current pixel-space blend still honors gradient flows because it only manipulates the final sample.

4 Code Walkthrough

The main entry point enumerates example cases and calls `run_masked_edit` with an image path, prompt, click coordinates, and output prefix. Inside this function we load the image via Pillow, resize to 512×512, and cache the SAM mask. The pipeline instantiation lives at the top of the script so that the heavy model load happens only once, and the scheduler is replaced with `EulerAncestralDiscreteScheduler` to match the expected sampling behavior.

Within the core loop we convert the mask into both 512×512 pixel space and a smaller latent resolution, which creates an avenue for future work that blends at intermediate timesteps. The result of the InstructPix2Pix call is stored under `res_baseline`, and the script writes four PNGs per example (‘`case_original`‘, ‘`case_mask`‘, etc.) into ‘results/‘. Because all I/O is centralized in this function, adding logging or metrics instrumentation is straightforward.

5 Data Augmentation and Mask Variation

The Segment Anything mask is not treated as sacred; we include helper code to dilate, blur, or jitter the mask if more forgiving edits are desired. For the current experiments we keep the mask binary, but the infrastructure is ready for soft masks by converting the binary mask to float tensors and applying Gaussian blurs. These augmentations can act as ablations when evaluating how sensitive the diffusion model is to mask accuracy.

The script also scales the mask from the original image resolution to the diffusion resolution, handling arbitrary input sizes. Because SAM runs at the original resolution, we compute the ratio and apply nearest-neighbor interpolation so the mask boundaries stay crisp. This means the plan can easily support new images without careful manual resizing.

Table 1: Canonical evaluation cases stored under `data/images/`.

Case	Instruction	Mask target
Shirt	“Change the shirt to bright red leather”	Shirt area
Dog	“Turn the dog into a playful robot”	Entire dog body
Car	“Make the car glow like a futuristic hovercraft”	Vehicle exterior

6 Comparison to Alternative Strategies

Earlier sections of the proposal considered retraining InstructPix2Pix with mask conditioning. That path would require synthetic dataset generation, LoRA fine-tuning, and multi-GPU compute, which goes beyond the current scope. Mask-Blended Inference, instead, leaves the diffusion weights untouched and imposes the mask as a post-processing constraint. This keeps the training cost at zero and makes it easy to experiment with different masks or instructions without longer training loops.

The downside is a reduced ability to influence intermediate denoising steps; we do not embed the mask inside the cross-attention layers. The upside is that the approach is training-free, so it can be implemented and evaluated within hours rather than days. For a final report, the baseline and mask-blended output comparisons directly highlight this trade-off.

In essence, mask blending sits between prompt engineering and training-time conditioning: it constrains the final sample with minimal engineering effort. This makes the technique especially suitable for coursework and rapid prototyping, where the goal is to show understanding of spatial conditioning without incurring the multi-day cost of dataset creation or solver reimplementation.

7 Experimental Setup, Dataset, and Results

Experiments focus on three representative images stored in `data/images/`. Each image includes a manual description of the desired edit so that the same instructions can be reproduced by anyone running the script; the instructions are intentionally concise yet descriptive (see Table 1). SAM receives a single click near the object center, and inference runs for 20 denoising steps with an `image_guidance_scale` of 1.5. After sampling, we store four images per example (original, SAM mask, baseline edit, blended edit) in `results/`, making it easy to compare the mask-guided output to the unconstrained baseline.

7.1 Hardware and Dependencies

All dependencies are documented in ‘README.md‘; the baseline command used earlier is:

```
python3 -m pip install torch torchvision torchaudio diffusers transformers accelerate scipy sa
```

This ensures the environment contains the heavy diffusion stack plus SAM. On the Mac host with no CUDA GPU, inference takes several minutes per example, so the plan is to run the script on a CUDA-equipped machine (local GPU or Colab) to finish the evaluations comfortably. The script automatically creates ‘results/‘ and logs messages via ‘print()‘ so the user can monitor progress.

8 Results and Qualitative Observations

Because this environment lacks a CUDA GPU, python inference is slow on the Mac host; a single example takes several minutes. That said, each case produces four PNGs in `results/`, and once generated we can visually compare the mask-guided edit to the baseline. The mask-blended output preserves background textures and only modifies the requested object, whereas the baseline often introduces stray artifacts outside the SAM region. These output pairs are ready to be captured in the final report as the required qualitative figures.

Each example includes metadata (prompt, click coordinates) that can be reused to replicate the result. The mask PNG communicates the spatial constraint, so the reviewer can quickly inspect where edits occurred. Generating the baseline edit alongside the masked edit also makes it clear how much the mask improves compositional fidelity—this comparison is the heart of the ‘Mask-Blended Inference’ argument.

9 Reproducibility and Submission

The README describes the dependency installation command (‘torch‘, ‘diffusers‘, ‘segment-anything‘, etc.), points to ‘requirements.txt‘ for reproducibility, specifies the SAM checkpoint placement (‘models/sam_vit_h_4b8939.pth‘), and lists the commands to run the pipeline (‘python3 submission/code/project_run.py‘) and the metric script (‘python3 submission/code/evaluate$_m$etrics.py‘).*Sampleimagesstayunder*data/images/, *checkpointsunder*models/, *andresults*

To regenerate the final report figures, simply delete the ‘results/‘ artifacts and rerun the script on a GPU machine; the outputs will land back in ‘results/‘ with predictable filenames (‘case_*‘). The README also reminds the user to download the SAM checkpoint manually from the official release page before running the script, ensuring no missing dependencies once the environment is recreated on another computer.

10 Model Versions and Licenses

The pipeline depends on public checkpoints: ‘timbrooks/instruct-pix2pix‘ from Hugging Face and ‘sam_vit_h_4b8939.pth‘ from Facebook Research. Both models are released under permissive licenses (MIT-style for InstructPix2Pix and Apache 2.0 for SAM), so the code and results can be shared with attribution. We log the exact versions in the README to avoid drift and ensure anyone reopening the project loads the same weights.

11 Implementation Issues and Troubleshooting

Several practical issues arose during development. First, PyTorch on macOS installs CPU-only binaries by default, so the pipeline crashes if run with CUDA-specific flags. Installing the standard ‘torch‘ package via ‘python3 -m pip install torch torchvision torchaudio‘ works, but for actual experiments the same command should run on a CUDA host to fetch GPU-enabled wheels. Second, SAM masks must be resized back to 512×512 after prediction to align with the diffusion output, so we aggressively use nearest-neighbor interpolation to keep the boundaries sharp.

Third, InstructPix2Pix consumes a lot of VRAM, so the script loads the pipeline once and reuses it for every case; this prevents repeated initialization from exceeding system memory. Logging statements before and after each major step help detect stalls or missing files (e.g., a missing SAM checkpoint). These observations will be included in the final README so others can troubleshoot runtime issues.

12 Detailed Example Cases

The three curated examples demonstrate the different categories of edits we care about.

1. **Shirt recoloring:** The original portrait has a neutral jacket, and the instruction ‘‘Change the shirt to bright red leather’’ requires precise control around the upper torso. SAM

isolates the jacket, and the blended edit recolors it while keeping the face and background untouched.

2. The mask is typically tight around the garment, so we do not rely on dilations or manual brush strokes. Post-hoc inspection of the mask PNG ensures the user-selected point falls onto the intended region.

3. **Dog transformation:** A puppy sits on a grassy meadow. The instruction ``Turn the dog into a playful robot'' should not affect the grass or sky. The mask-guided edit keeps the environment static while allowing metallic textures to appear over the dog's body.

4. We expect the mask to cover the entire dog silhouette because the click targets the center of mass; this makes the edit appear as a localized robot overlay.

5. **Car restyling:** A street scene with a car is retargeted to ``Make the car glow like a futuristic hovercraft.'' SAM isolates the vehicle's body so the street, pedestrians, and skyline remain in their original state, while the car itself acquires neon highlights.

6. This case also shows how mask blending prevents the diffusion model from altering the pavement or introducing lighting changes outside the vehicle contour.

13 Case-Specific Observations

Running all three canonical cases revealed a few notable behaviors: the shirt edit often benefits from slight increases in image_guidance_scale, the dog edit tends to saturate faster, and the car edit occasionally pushes neon highlights into shadows. Documenting these observations helps tune future experiments and provides qualitative bullet points for the final report. We also log the click coordinates next to each output so the same setup can be repeated precisely.

14 Evaluation Plan

Beyond the qualitative comparisons, the plan includes a lightweight evaluation suite:

- **Spatial precision:** Compute the absolute difference between the baseline edit and the original image, threshold that change map, and intersect it with the SAM mask to obtain mIoU scores that reflect how well edits stay within the target region.
- **Instruction fidelity:** Use CLIP similarity between the edited image and the instruction text, and optionally compare ranks when scoring a batch of edits against multiple prompts.
- **Human study template:** Prepare a short survey where participants rate whether the edit stayed inside the mask and whether it reflects the instruction. With 10{15 raters, report mean Likert scores and inter-rater agreement.

These metrics complement the qualitative figures. The mIoU provides a hard quantitative signal for spatial control, CLIP scores capture semantic compliance with the instruction, and the human study documents perceived fidelity. Together, they satisfy the assignment's request for both automatic metrics and user-facing evaluation.

The repository ships ‘submission/code/evaluate$_m etrics.py‘, which scans‘results/‘,$

15 Timeline and Risk Mitigation

The original proposal anticipated a four-week timeline, and the current implementation follows this cadence: Week 1 covered literature review and baseline setup, Week 2 built the SAM pipeline and data plumbing, Week 3 developed the mask-blended integration, and Week 4 focuses on evaluation artifacts and the final report. Because inference on this machine is CPU-bound, the timeline now includes an extra step for rerunning the script on GPU within Week 4 so the PNG outputs are ready for documentation.

Primary risks include SAM missegmentations (mitigated by reviewing masks for the chosen cases), GPU time constraints (mitigated by planning for a Colab or remote GPU run), and documentation completeness (mitigated by this report, README, and the repo organization). Each risk maps to a concrete mitigation action so the final deliverables remain on-track.

- **Week 1:** Read papers and reproduce baseline InstructPix2Pix figures.
- **Week 2:** Implement SAM-based mask generation and data flow.
- **Week 3:** Add mask blending to the routine, validate on the three canonical cases.
- **Week 4:** Run GPU inference to collect results, finalize report, and package submission.

16 Discussion and Analysis

The mask-blended inference strategy effectively keeps non-target regions static without retraining, which makes it ideal for tight deadlines and limited compute budgets. Anyone can swap masks, instructions, or images instantly, and the deterministic blending step keeps the workflow easy to inspect because each output can be traced back to a specific mask.

16.1 Limitations

Because the blending happens after the full diffusion pass, intermediate timesteps still contain unmasked noise. In other words, we constrain the result but not the trajectory, so the generator may still spend capacity modeling the entire scene before we crop the background back to the original. This approach also relies on SAM producing accurate masks from a single click, which can fail on cluttered scenes or when the click falls outside the object. Finally, the current implementation runs on CPU, so additional GPU resources are required to finish the evaluation suite in a practical timeframe.

16.2 Quantitative Metrics

Once the outputs are generated on a GPU machine, we can compute a simple change map by taking the absolute pixel difference between the baseline edit and the original image, binarizing that map, and comparing it to the SAM mask to yield a mean intersection-over-union (mIoU) score. Instruction fidelity can be approximated by a CLIP similarity score between the edited image and the textual instruction, or by ranking multiple edits with a captioning model. Because the blending is deterministic, these comparisons will be reproducible and scriptable.

17 Appendix: Running the Pipeline

To run the script from scratch:

1. Install all dependencies ('torch', 'diffusers', 'segment-anything', etc.) and download 'models/sam_vit_h_4b8939.pth'.
2. Place your test images inside 'data/images/' and update the prompt/click coordinates in 'submission/code/project_run.py' as needed.
3. Execute 'python3 submission/code/project_run.py'. The script prints progress for each case and writes 'results/' artifacts.
4. Collect the PNGs for each example (e.g., case_man_shirt_original.png, case_man_shirt_mask.png, case_man_shirt_baseline.png, case_man_shirt_ours.png) for the final report or README.

This appendix ensures reproducibility without guessing which files to edit or where outputs should appear.

18 Command Reference

The following commands capture the key steps from environment setup through result generation.

Table 2: Command reference for the mask-blended pipeline.

Purpose	Command
Install diffusion stack	`python3 -m pip install torch torch`
Run inference script	`python3 submission/code/project_ru`
Clean results	`rm results/case_*_*.png`
Inspect latest outputs	`ls -1 results/case_*_ours.png`

Keeping this table updated helps the next person reproduce the work without scanning the README.

19 Metric Implementation Notes

We compute the mIoU by thresholding the absolute difference between the baseline edit and the original image: $D = |y_{\texttt{baseline}} - x_0|$. Thresholding D at τ yields the binary change map C, and intersection over union with the SAM mask gives the spatial precision score. For the CLIP score we use the official model to encode both the edited image and the instruction, then compute cosine similarity. These artifacts are logged per example.

The CSV-based human study template pairs each case with two Likert ratings (spatial compliance, instruction fidelity), which can be aggregated with Pandas so the final report includes mean scores and standard deviations. The metric implementation notes will be shipped with the code so future reviewers can reproduce the evaluations without re-deriving formulas.

20 Human Study Protocol

The planned human study is small (10{15 participants) but focused on two axes: spatial compliance and instruction fidelity. Each participant sees paired images (baseline vs. mask-blended) for the three canonical cases and answers two Likert questions per pair. This yields mean ratings and allows for computing inter-rater agreement.

Participants will receive a short instruction sheet explaining which images to compare and what ``staying inside the mask'' and ``obeying the instruction'' mean. The study randomizes the presentation order to minimize bias, and the results can be summarized as mean Likert scores in the final report alongside a few representative comments.

21 Future Extensions

Beyond the current mask blending, future versions could inject the mask into the cross-attention layers of the diffusion U-Net or explore latent-space compositing at each timestep. Integrating user-provided bounding boxes or scribbles with SAM would also refine the masks, and the current codebase already accepts alternative mask formats thanks to the modular mask loader.

We also plan to expand the automated metric suite so each run logs per-example mIoU, CLIP score, and runtime. Automating 'results/' generation across more images will further demonstrate reproducibility while keeping the solution training-free.

22 Acknowledgments

Thanks to the CS 5404 teaching team for the detailed final-project instructions and for pointing to the public checkpoints that made this pipeline possible. The SAM and InstructPix2Pix weights are both publicly hosted, which greatly simplifies dependency management.

23 Deployment and Packaging

The repository is organized so the 'submission/' directory alone contains the artifacts that get submitted: proposal PDF, final report PDF, and runnable code. The rest of the workspace holds supporting data ('data/'), models ('models/'), and generated outputs ('results/'). When preparing a submission archive, simply zip 'submission/' and ensure 'README.md' communicates the extra assets. This division keeps the deliverable bundle lightweight while the rest of the repo retains reproducibility assets.

24 Conclusion and Future Work

Mask-Blended Inference fulfills the proposal promise by using SAM masks to guide InstructPix2Pix at inference time without changing the pretrained weights. The reorganized repository keeps the submission bundle, data, models, and results clearly separated, which simplifies sharing and grading. In future work I plan to instrument the pipeline with automated metrics (mIoU, CLIP score), add a lightweight ablation over mask noise or dilation, and explore more advanced blending strategies that operate in latent space between denoising steps. A small human evaluation could also quantify whether viewers perceive the edits as precise and localized.

Code Availability

Code lives in submission/code/project_run.py (masked inference) and can be run after installing the dependencies listed in README.md. The models/sam_vit_h_4b8939.pth checkpoint and the sample images under data/images/ are also required.

References

[1] Tim Brooks, Aleksander Holynski, and Alexei Efros. InstructPix2Pix: Learning to follow image editing instructions. CVPR 2023.

[2] Alexander Kirillov et al. Segment Anything. ICCV 2023.