A. Frequently Asked Questions (F.A.Q.)

1. **Why is the independent test set so small, and is it sufficient to evaluate generalization?**
   The independent test set contains 121 images that were captured by non-authors but annotated by the primary author. Its current size is primarily a consequence of how many suitable contributions we received so far; we would prefer a larger test set and are actively seeking to grow it. In parallel, we are experimenting with additional feces-related image collections hosted on public platforms (e.g., community datasets on sites such as Roboflow). These external datasets were not part of the original design of ScatSpotter, and they raise separate licensing, attribution, and quality-control questions, so they are not included in the core benchmark or main tables in this paper.
   The current contributor test set is strictly photographer-disjoint from the main author data and is meant to provide an initial, realistic check on *cross-photographer* generalization, while the larger author-collected data drive training and validation. However, due to its limited size, it should be viewed as an informative but limited indicator of generalization across larger geographic distribution shifts (*e.g.* the dataset does not contain many examples of sandy environments).

2. **Why focus on a single class (dog feces)? Is this problem non-trivial?**
   The benchmark is intentionally single-class. We prioritize high-quality annotations for one difficult, under-served class instead of many shallow labels. Dog feces are extremely common in real-world settings (parks, sidewalks, trails) but often *not* visually obvious, especially in cluttered foliage, snow, or low light. In practice, even attentive owners sometimes lose track of where their dog went, and visually locating poop is a genuine nuisance task.
   Although the topic is niche, the computer-vision problem is not trivial: the class is small, highly imbalanced, and strongly confounded by leaves, sticks, rocks, mud, stains, and other background clutter. Learning to solve this requires fine-grained texture and shape cues that are also relevant for broader bio-waste detection, environmental contamination, and aspects of animal behavior monitoring.

3. **Given the narrow, single-class focus and limited geographic diversity, how far can results on ScatSpotter be generalized?**
   We do not claim ScatSpotter is globally representative. Training and validation data are dominated by a single photographer, a small set of dogs, and a specific geographic region. The contributor test set partially offsets this by introducing different users and capture styles, but the overall benchmark remains biased toward the author's environment and devices. We view ScatSpotter as a focused, well-annotated base domain that can be extended with additional classes (e.g., other waste types) and combined with other datasets for new cities, species, or devices. As with any dataset, careful evaluation on the target domain remains important.

4. **How were the splits made (train/validation/test), and how does this affect overfitting and generalization?**
   Initially, the validation set consisted of the earliest batch of collected images, with the remainder used for training. This introduced temporal and domain shift by design. As the dataset grew, we found the validation set too small, so we augmented it with images sampled periodically (e.g., every third day in a given year). This increased its size and eliminated the chance that images from the same walk appear in both train and validation, which could artificially inflate performance.
   The independent test set is composed of contributor images only and does not include negative-only or "after" frames. This setup encourages models to generalize across photographers and capture styles rather than overfit to the main author's habits and devices.

5. **What exactly is the "before/after/negative" (BAN) protocol, and how are these triplets represented? Were multiple viewpoints of the same scene captured?**
   Many examples are captured as short sequences: a *before* image (poop present), an *after* image (poop removed, with a best-effort similar viewpoint), and sometimes a later *negative* image of the same area. This began as a practical annotation aid: flipping between images helps the annotator spot small or camouflaged instances. It also naturally produces multiple viewpoints and time offsets of the same scene, which are valuable for studying hard negatives, change detection, and contrastive objectives.
   In our code, candidate BAN groups are identified using temporal proximity and image matching (e.g., SIFT-based alignment) and are tagged accordingly. In current releases, BAN roles are derived by the provided scripts rather than stored as static fields in the annotation file; future versions will expose these roles as explicit metadata so downstream work can more easily leverage them.
   In our code, candidate BAN groups are identified using temporal proximity and image matching (e.g., SIFT-based alignment) and are tagged accordingly. In current releases, BAN roles are derived by the provided scripts rather than stored as static fields in the annotation file; future versions will expose these roles as explicit metadata so downstream work can more easily leverage them.
   More generally, images with no polygons labeled "poop" are implicitly negative examples, but the current version does not make them explicit.

6. **How were annotations produced and validated?**
   Most polygon annotations were drawn and reviewed by

a single annotator (the primary author). In some cases, detector outputs (e.g., from YOLO or MaskRCNN) provided initial proposals that were then edited, but we do not store explicit flags indicating which polygons were model-seeded. All annotations pass through the same human review process. When the annotator cannot confidently determine whether a region is poop (e.g., old stains, heavily decomposed material), it is labeled as "unknown" or "ignore" rather than forced into positive or background. The independent contributor test set is annotated by non-authors; systematic differences in their style give a realistic view of how models handle non-author imagery.

7. **What else is in the images?**

   Although the benchmark focuses on dog feces, the scenes contain a wide variety of natural clutter and small objects, including leaves, sticks, rocks, pine cones, helicopter seeds, tree bark, grass patches, dirt, occasional micro-trash (e.g., bottle caps, cigarette butts), and other incidental elements (e.g., shadows, roots, mulch, acorns, mice). Some of these appear as sparse labels (often when they caused false positives or were otherwise interesting), but they are not exhaustively annotated across the dataset. Many poop instances also carry free-form description tags (e.g., `fresh`, `old`, `crumbly`, `diarrhea`, `messy`, `occluded`, `camoflauged`, `snowcovered`, `unknown`/`unsure`). For some interesting cases, additional text tags were added to annotations with the idea that in the future VLMs could use them as truth anchors and help propagate text labels to other similar cases (*e.g.* "old", "fresh", "sick"). These auxiliary tags are potentially useful signals, but they are not yet standardized enough to define separate benchmark tasks. These extra labels are excluded from stats reported in the main text.

8. **What is the primary task in this benchmark (detection or segmentation), and how should I interpret the baseline models and metrics?**
   The primary task is *object detection*: find and localize poop instances in an image. We annotate polygons for each instance and derive bounding boxes from these masks for use with commonly used detectors. The dense masks also support segmentation models and more detailed error analysis, but segmentation is secondary to detection in the current benchmark.
   The baseline suite is intentionally heterogeneous: it includes commonly used models in practice (e.g., YOLO-style one-stage detectors, MaskRCNN-style two-stage detectors), segmentation networks (e.g., ViT-based), and open-vocabulary/foundation detectors (e.g., GroundingDINO), each in a configuration that is standard and well supported for that architecture. Different models therefore run at different input resolutions and capacities. We do not claim these baselines are fully optimized or perfectly comparable on every axis; they should be treated as reasonable reference points, not a definitive ranking.
   We report detection metrics such as AP and AUC, and also F1, IoU, recall/TPR, and precision, which are important in this highly imbalanced regime. For thresholded metrics we select a single global threshold per model on the validation set and reuse it on the test set, unless otherwise noted. Pixel-level metrics are only reported for architectures that produce masks; detector-only models (e.g., some YOLO variants, GroundingDINO in its standard form) do not output per-pixel predictions.

9. **Why include relatively few baseline architectures, and why not emphasize more lightweight or mobile models?**
   In an ideal world, we would run and maintain a large zoo of models at every evolution of the dataset. In practice, the current ecosystem for robust, large-scale benchmarking of object detection does not yet provide easy, off-the-shelf solutions for doing this in a maintainable way. Our effort in this work was focused primarily on building, curating, and documenting the dataset. As a result, we select a small, diverse set of baselines that cover key families (two-stage detectors, one-stage YOLO-style models, dense segmenters, foundation/open-vocabulary detectors). Some of these are reasonably lightweight and relevant for on-device or robotic deployment; others are heavier and are used both as strong baselines and as annotation assistants. We would like to expand the benchmark suite as better, less ad-hoc benchmarking frameworks become available, and we place relatively less weight on the specific baseline roster in this initial dataset paper.

10. **What is the real benefit of content-addressable distribution and hash-verifiable data here?**
    Traditional dataset hosting often relies on a stable URL whose contents may change silently over time, requiring implicit trust in the host. In contrast, content-addressable storage (e.g., IPFS CIDs, torrent magnet hashes, hashed annotation files) ties each version of the dataset to a cryptographic digest of its exact contents. If the bits change, the identifier changes. This has two main benefits:
    (a) It becomes easy to verify that your local copy matches the one used in a given paper, by checking the published hashes and simple dataset statistics.
    (b) It becomes hard to accidentally or silently modify the dataset without producing a new identifier, which helps keep future extensions and bug fixes honest and traceable.
    We still provide user-friendly access via platforms such as Hugging Face, but we encourage scripts and papers to

reference the content hashes so that experiments remain reproducible even if hosting providers or URLs change.

11. **What privacy considerations apply to this dataset, including EXIF metadata?**
Some images in the dataset include EXIF metadata (e.g., camera parameters and timestamps). A subset of images—primarily those captured by the author—include GPS location data in their original form. In cases with privacy concerns, we selectively remove sensitive EXIF fields prior to public release.
When consent is obtained, encrypted artifacts are distributed. These allow for rehydration of the original metadata by trusted parties with access to the decryption key. Because dataset snapshots are distributed via content-addressed and peer-to-peer mechanisms (*e.g.* IPFS and BitTorrent), users should treat published snapshots as immutable.

B. Dataset

B.1. Additional Comparisons

In Section 2 we compared to related work. Here we expand on this by comparing our analysis plots. Every dataset is converted into the COCO format and visualized using the same logic. Figure 2 visualizes the annotations of all datasets. We make similar visualizations for other comparable dataset metrics. Figure 7 shows the number of annotations per image. Figure 8 shows of image sizes in each dataset. Figure 9 shows the distribution of width and heights of oriented bounding boxes fit to annotation polygons. Figure 10 shows the area of each polygon versus the number of vertices (which could be used to estimate the likelihood a polygon was generated by AI for our dataset). Figure 11 shows the distribution of centroid positions (relative to the image size).

B.2. Additional Information

In Section 3 we provided an overview of several dataset statistics. In this appendix we expand on that with additional plots. The distribution of image pixel intensities is illustrated in Figure 12. The distribution of images collected over time is shown in Figure 13. The distribution of annotation location is shown in Figure 14 and sizes is shown in Figure 15 and Figure 16.

C. Data Distribution & Transfer

In Section 5 we briefly presented a brief set of data distribution experiments. Here, we provide more background detail, motivation, and discussion.

Empirical evidence suggests that a substantial proportion of scientific studies have low reproducibility rates, which has raised concerns across various disciplines [2]. Ideally, scientific research should be independently reproducible. Despite higher success rates in computer science (up to 60%) compared to other fields, there is still room for improvement [11, 14, 47]. Addressing this issue requires not just better experimental documentation but also more reliable and accessible data distribution methods. Specifically, this involves robustly codifying data download and preparation processes.

Centralized data distribution methods allow for codified data access by storing URLs that point to datasets within the code, offering fast and direct access. However, this approach lacks robustness. It can fail if the provider goes offline, changes the URL, or stops hosting the data. Additionally, cloud storage can be expensive, and users must trust that the provider delivers the correct data — a risk that can be mitigated by using checksums to verify data integrity.

In contrast, decentralized methods allow users to access data in the same way, even if the organization hosting the data changes. By leveraging content-addressable storage, where the dataset checksum acts as both the key to locate and validate the data, these methods ensure data integrity and nearly eliminate the risk of dead URLs, provided that at least one peer retains the data. While decentralized systems face challenges such as longer connection times, increased network overhead, and the need for a robust peer network, their ability to ensure data access via a static address motivates our investigation

Specifically, we focus on two prominent candidates: BitTorrent and IPFS. BitTorrent [8, 9] is a well known sharing protocol that originally relied on centralized trackers and databases of torrent files to connect peers. While trackers and torrent files are still prominent, torrents can be published to a distributed hash table (DHT) using the Kademlia algorithm [37]. This makes it an strong candidate for a decentralized distribution mechanism. On the other hand, IPFS (InterPlanetary File System) [4, 6] is a newer tool directly build directly on a DHT. IPFS has been likened to "a single BitTorrent swarm, exchanging objects within one Git repository". Both IPFS and BitTorrent are content addressable at the dataset level, which makes them both appropriate for our use case where we seek a static address that can be used to robustly access data.

It is worth noting that git-based [7] systems like HuggingFace [32] with large file storage do gain some decentralized properties via multiple remotes, but not content identifiers.

For practitioners, key concerns are how quickly and reliably data can be accessed. By comparing decentralized and centralized mechanisms access times for our dataset, we aim to make explicit the tradeoffs between the methods and inform decisions on adopting an approach.

C.1. Data Distribution Discussion

To assess the effectiveness of each mechanism we programmatically download our 42GB dataset and measure the time

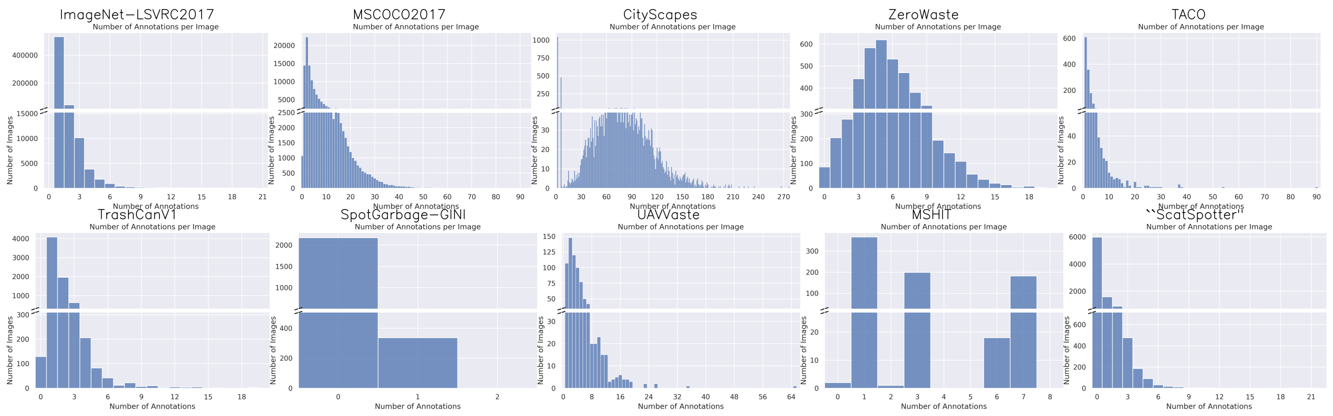


Figure 7. Number of annotations per image in each dataset.

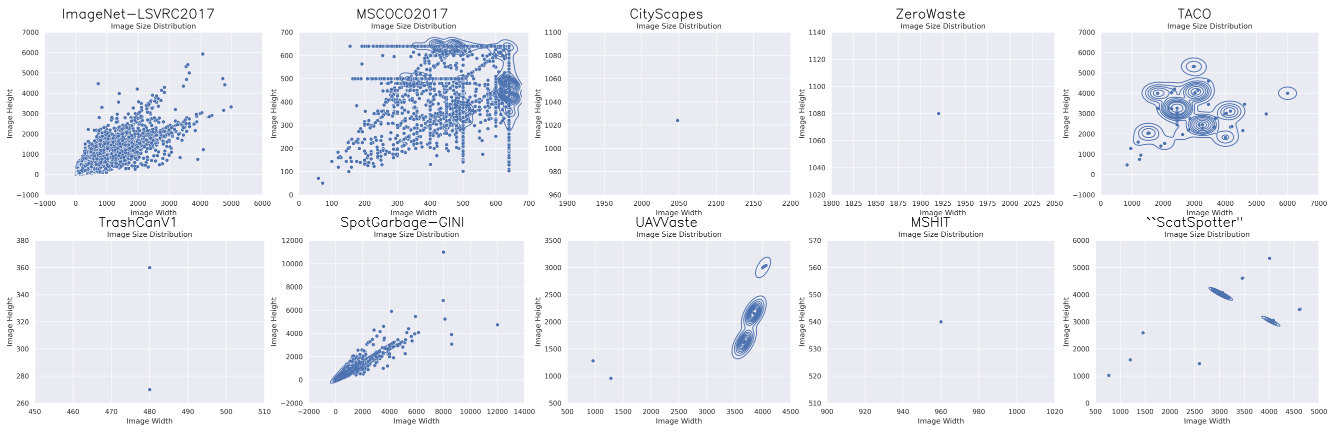


Figure 8. Image size distributions of each dataset. Ours has two primary width/heights.

required to complete the transfer. Each experiment was run five times, machines we controlled were separated by $\sim$30 kilometers with an average ping time of 48.48 ms. For each test, we log transfer start and end times along with notes and code (provided in code repo).

While our measurements provide a reasonable estimate of for access time for each mechanism, there are notable limitations in our methodology. First, different machines and networks have different upload and download speeds, and network congestion is variable. For decentralized methods, we lack an automated mechanism separate peer-connection time and actual download time. Additionally, Girder and HuggingFace required data to be packed into compressed archives, improving transfer efficiency due to fewer file boundaries. In decentralized cases, we provide granular access to each file in the dataset, which avoids an extra unpacking step and enables sharing of the same file between different versions of the datasets and simpler updates, but decreases transfer efficiency. Due to this, we provide both a compressed and uncompressed rsync baseline. Another confounding factor is that with decentralized mechanisms the number of seeders is not controlled for. Subsets of the data have been hosted on IPFS for years, and portions of the dataset may be provided by unknown members of the network. For BitTorrent, our initial transfers only had one seeder, but during our tests other nodes accessed and started to provide the data.

Despite significant testing limitations, our measurements quantify the expected data-access time penalty to gain the advantages of decentralized mechanisms. With these limitations acknowledged, we present the transfer times statistics in Table 3. Alongside these measurements, several observations are worth noting. Transferring files using IPFS had significantly delayed peer discovery times, and we were only able to connect two machines after manually informing them of each other's peer ID. For BitTorrent, were unable to use the mainline DHT and fell back to using trackers. We believe these peer discovery issues are because the dataset has a small number of seeders. To test this, we downloaded other established datasets via IPFS and BitTorrent and found that the peer discovery time was almost immediate, suggesting that this becomes less of an issue as a dataset is shared. How-

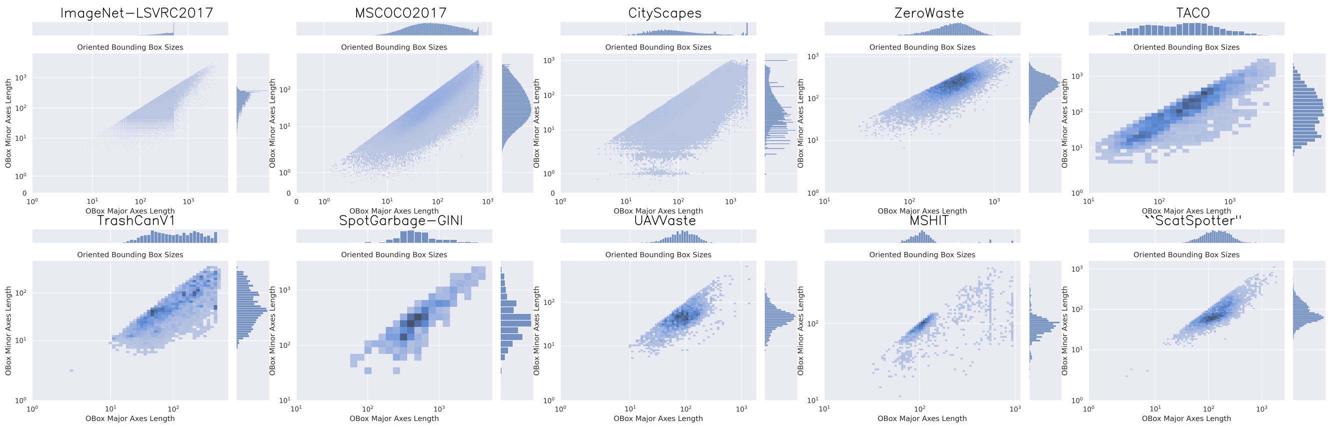


Figure 9. Oriented bounding box size distributions (log10 scale) of each dataset.

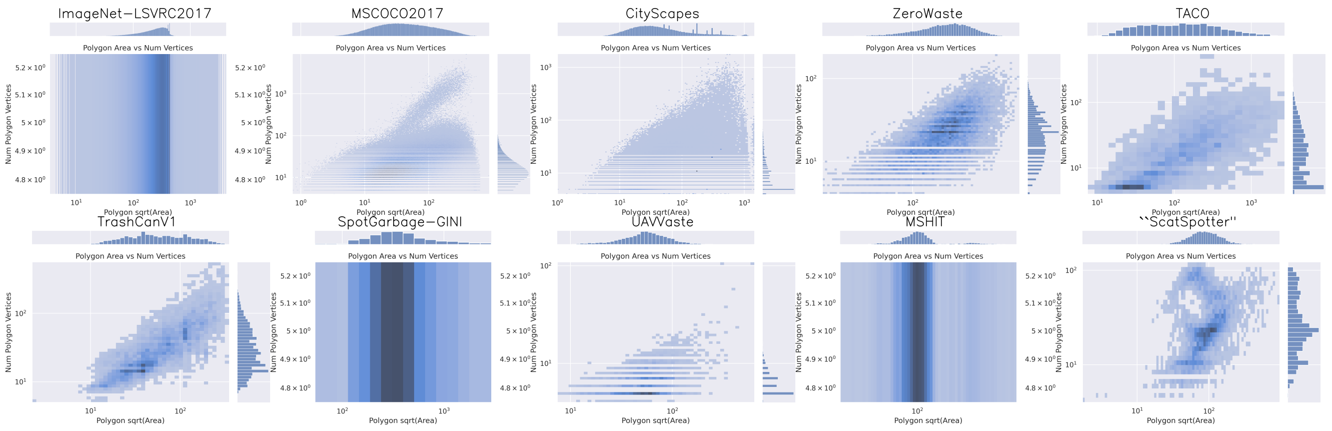


Figure 10. Polygon area versus number of vertices (log10 scale) for each dataset. The polygons with more vertices are more likely to be AI generated.

ever, the inability to quickly find a nearby peer is a major issue for initial or private dataset development.

C.2. Dataset Versions

An advantage of content identifiers is that they are resistant to link rot as long as at least one peer hosts the data, and more importantly, they can never resolve to the wrong content. This makes them highly attractive for scientific reproducibility. In this work we relied on two main dataset versions, each specified by a stable content-based identifiers:

Version from 2024-07-03

- IPFS CID: `bafybeiedwp2zvmdyb2c2axrcl455xfbv2mgdbhgkc3dile4dftiimwth2y`
- BitTorrent: `magnet:?xt=urn:btih:ee8d2c87a39ea9bfe48bef7eb4ca12eb68852c49`

Version from 2025-04-20

- IPFS CID: `bafybeia2uv3ea3aoz27ytiwbyudrjzblfuen47hm6tyfrjt6dgf6iadta4`
- BitTorrent: `magnet:?xt=urn:btih:27a2512ae93298f75544be6d2d629dfb186f86cf`

Note: the hash suffix of the magnet URL can be searched on `academictorrents.com`.

At the time of writing, the version of the dataset on HuggingFace is the latest, and we use git tags that correspond with the date of release and the IPFS CID to help identify dataset versions. However, unlike the decentralized methods, these are not guaranteed to point to the expected version of the dataset. At the time of writing the HuggingFace URL is: `https://huggingface.co/datasets/erotemic/scatspotter` and the Girder URL is: `https://data.kitware.com/?#user/598a19658d777f7d33e9c18b/folder/66b6bc7ef87a980650f41f98`.

D. Model & Training Details

In Section 4 we provided our main results. However, a key limitation of these results is the imbalance between model types, with 42 of 47 trained models being VIT-ssegs,

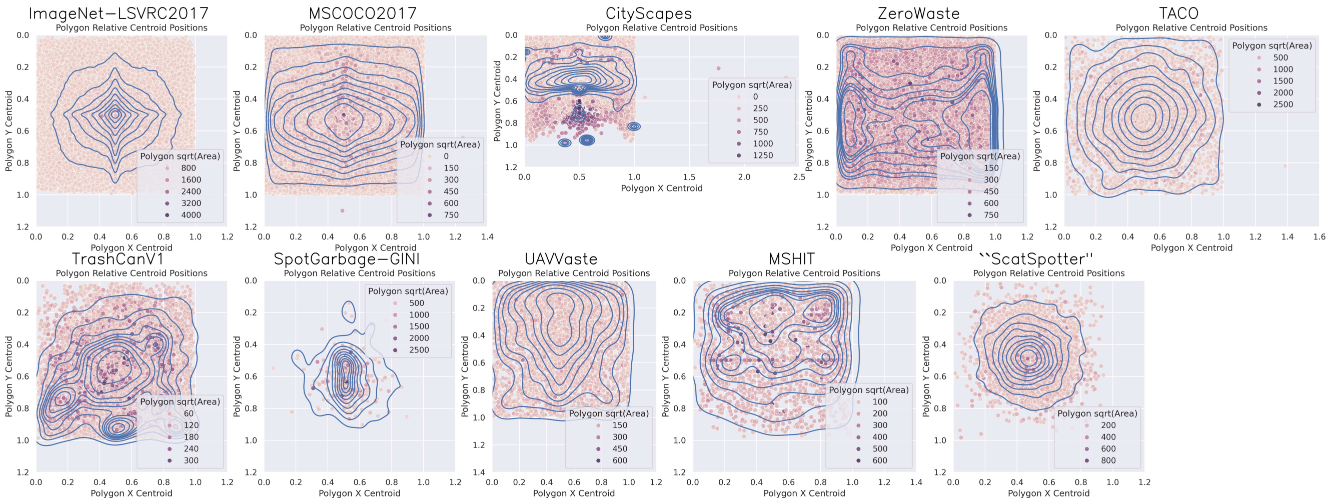


Figure 11. Polygon centroid relative distribution for each dataset. It is interesting to note patterns in this data. For instance, the outline of a street can be seen in CityScapes. In Zero Waste you can see the conveyor belt. ImageNet is more uniform. Ours is Gaussian distributed.

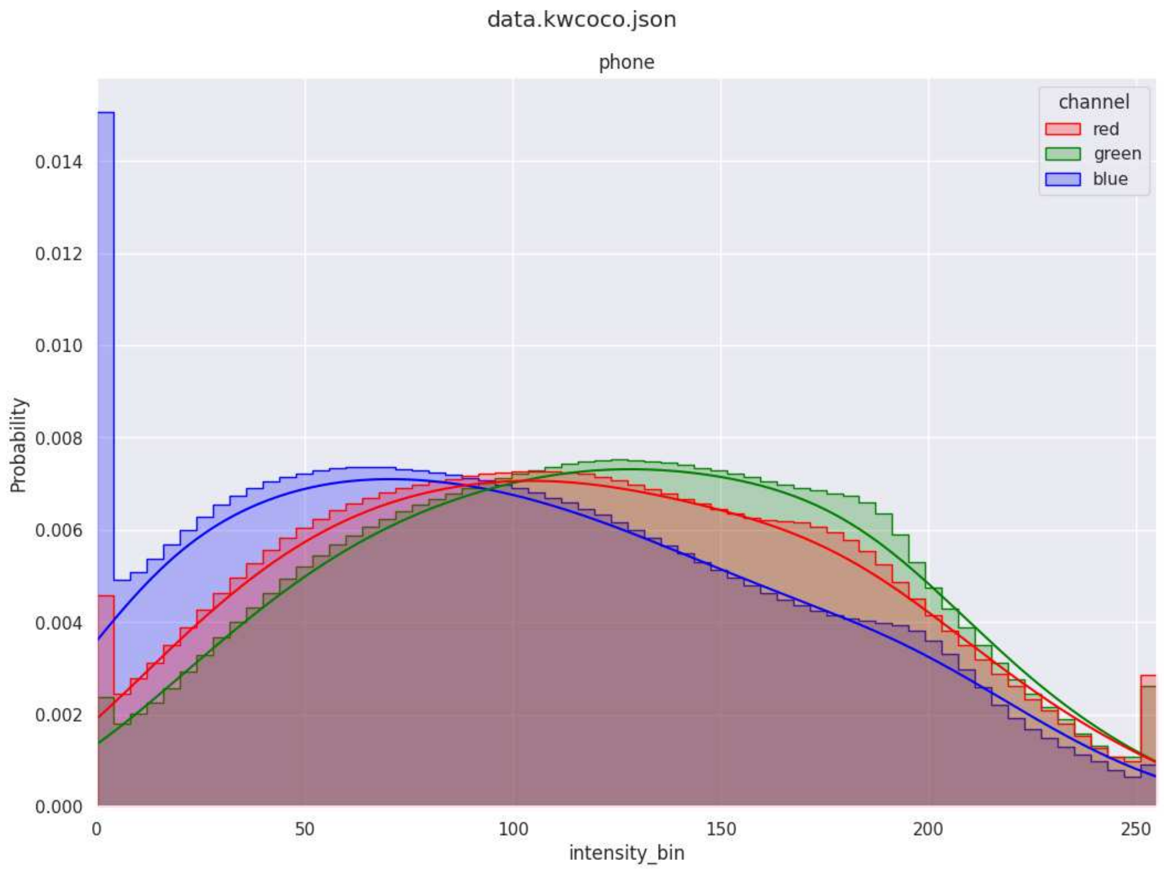


Figure 12. The “spectra” or histogram of the pixel intensities in the dataset. The dataset RGB mean/std is $[117, 124, 100]$, $[61, 59, 63]$. High and low saturated values occur, but are included in the stats. This was run on the older 2024-07-03 snapshot.

2 MaskRCNN models, 2 YOLO models, and 1 tuned GroundingDino model. Future work could further optimize MaskR-CNN, Grounding DINO, YOLO, and other models to improve both performance and comparability, but these results

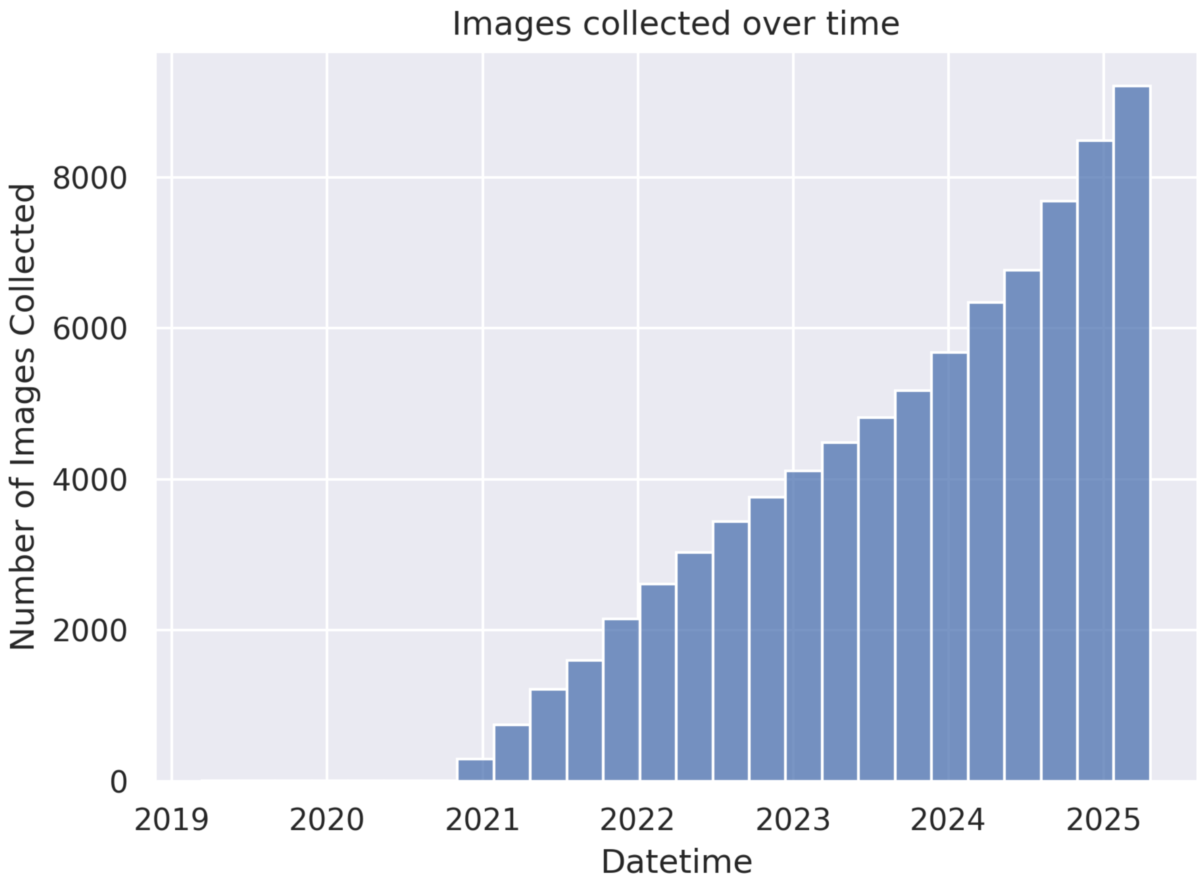
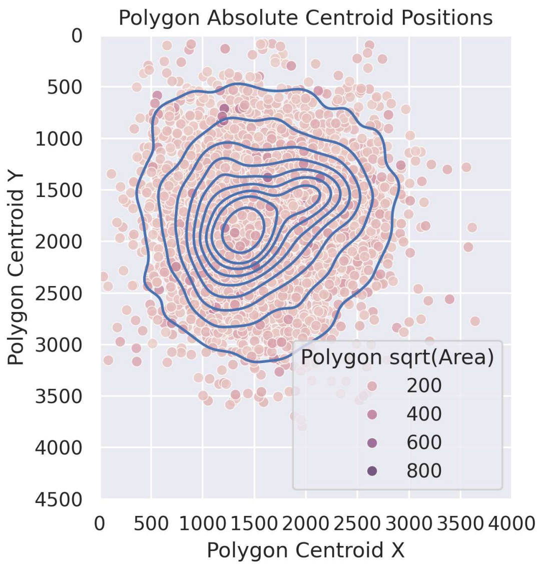
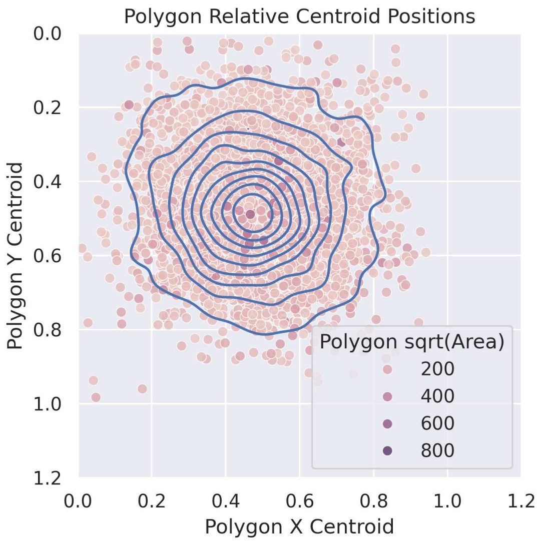


Figure 13. The number of images collected over time.

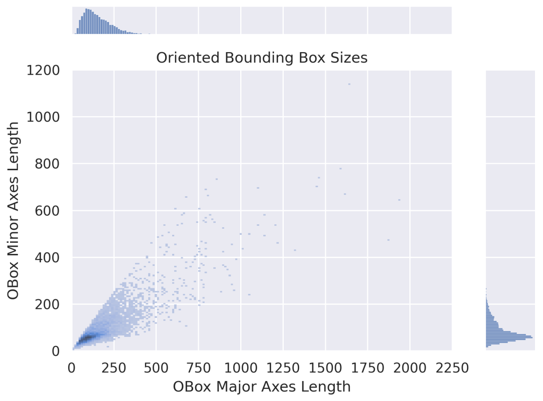


(a) Absolute pixel coordinates.

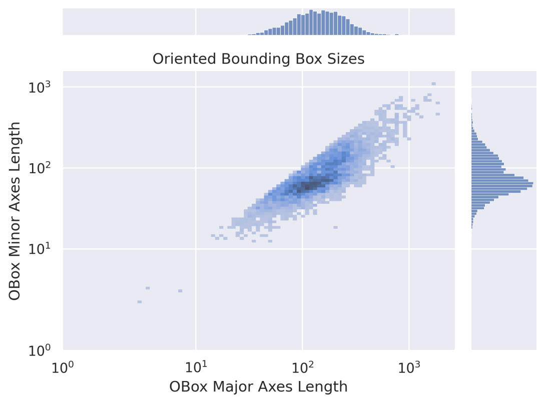


(b) Relative image coordinates.

Figure 14. The distribution of annotation centroids in terms of (a) absolute image coordinates and (b) relative image coordinates. The absolute centroid distribution is bimodal because some images are taken in landscape mode and other in portrait mode.



(a) Linear scale.



(b) Log10 scale.

Figure 15. The distribution of annotation sizes as measured by an oriented bounding box fit to each polygon. (a) shows this plot on a linear scale and (b) show this plot on a log scale.

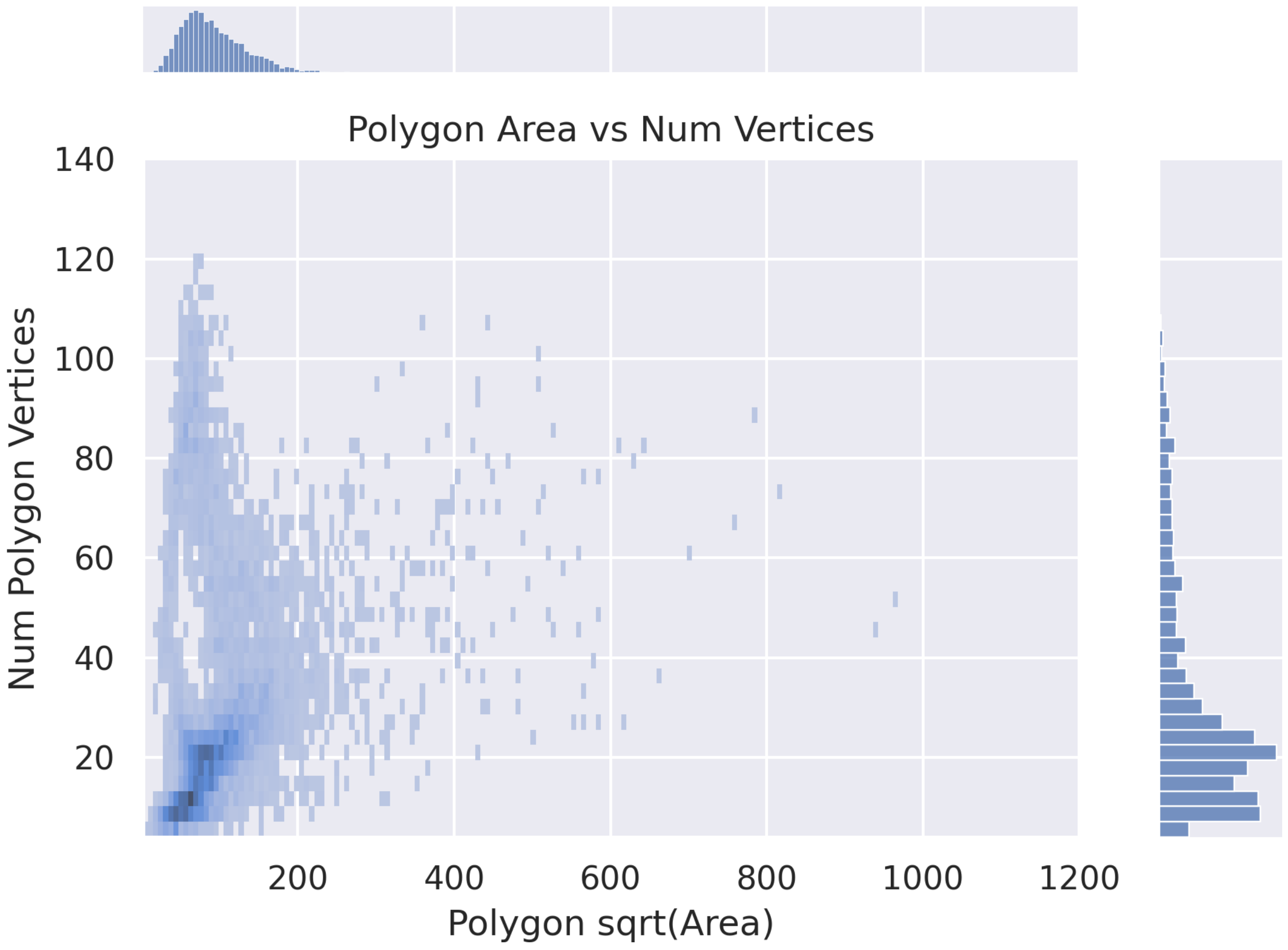


Figure 16. The distribution of polygon areas versus the number of vertices in the polygon boundary. The SAM model tends to produce polygons with a higher number of vertices than manually drawn ones. For smaller polygons there are two peaks in the number of vertices histograms likely corresponding to pure-manual versus AI-assisted annotations.

are enough to establish a useful baseline.

For non-VIT models we adhered as closely as possible to the default parameters of their respective frameworks, applying changes needed to support our generalized KWCoco format and to fit on a single GPU. For complete details, we provide links to the training and evaluation scripts for each model family (see below). A docker image with dependencies pre-installed is also available: `erotemic/shitspotter:latest`.

Experiment scripts (by task).

- **Grounding DINO:** `./experiments/grounding-dino-experiments`
- **YOLO-v9:** `./experiments/yolo-experiments`
- **MaskRCNN:** `./experiments/detectron2-experiments`
- **VIT-sseg:** `./experiments/geowatch-experiments`

D.1. Grounding Dino

GroundingDINO was evaluated in two modes. The zero-shot setting used the IDEA-Research/grounding-dino-tiny model from HuggingFace, applied directly to our validation and test splits with a fixed set of 10 prompts. The tuned variant used the community longzw1997/Open-GroundingDino implementation, initialized from `groundingdino_swint_ogc.pth` with a BERT text encoder on a single GPU. Training data was converted to ODVG JSONL format, and the label set reduced to two classes ("poop" and "unknown").

The preprocessing pipeline resized the shorter side of each image to 800 pixels while maintaining aspect ratio, corresponding to a median scale factor of $\sim$0.26 for our dataset.

In the zero-shot evaluation, we tested ten prompts and selected "animalfeces" based on the highest validation Box-AP. Table 4 shows the full ablation, illustrating that prompt choice strongly affects performance. Prompt choice has a large effect, and the best prompt differs between validation and test splits, but overall zero-shot results remain low.

D.2. YOLO-v9

YOLO-v9 experiments were based on the community WongKinYiu/yolov9 implementation. We trained both pretrained (ImageNet-initialized) and from-scratch variants, using our fork adapted for KWCoco input.

All images were resized to 640$\times$640, corresponding to a median scale factor of $\sim$0.16. Training used a batch size of 16 with batch accumulation set to 50, for an effective batch size of 800. Optimization used AdamW with learning rate 3×10^{-4} and weight decay of 0.01. The pretrained runs started from `v9-c.pt`.

D.3. MaskRCNN

MaskRCNN experiments were run using a Detectron2 fork with KWCoco support. Both pretrained and from-scratch models used the standard `R_50_FPN_3x.yaml` configuration, differing only in initialization: pretrained models used `detectron2://ImageNetPretrained/MSRA/R-50.pkl`, while the from-scratch models started randomly.

To fit training on a single GPU, we reduced the learning rate to 2.5×10^{-4}, set the batch size to 2, and trained for a maximum of 120,000 iterations. The maximum image dimension was capped at 1024, giving a median scale factor of $\sim$0.25 for our dataset.

D.4. VIT-sseg

This subsection provides additional details on VIT-sseg models, which were the first architecture we explored for this problem and therefore have the most extensive analysis compared to other networks.

To train VIT-sseg models we use the training, prediction, and evaluation system presented in [13, 20], which utilizes polygon annotations to train a pixelwise binary segmentation model.

In all experiments, we use half-resolution images, which means most images have an effective width $\times$ height of 2,016 $\times$ 1,512. We employ a spatial window size of 416 $\times$ 416 for network inputs, which means that multiple windows are needed to predict on entire images. During prediction, we apply a window overlap of 0.3 with feathered stitching to prevent boundary artifacts.

To address the class imbalance in our dataset (where positives are patches containing annotations and negatives contain no annotations), we adopt a balanced sampling strategy. Each "epoch" consists of randomly sampling 32,768 patches from the dataset with replacement, ensuring roughly equal numbers of positive and negative samples. We train each network for 163,840 gradient steps. For data augmentation we use random crops and flips.

Our baseline architecture is a variant [5, 20] of a vision-transformer [16]. The model is a 12-layer encoder backbone with 384 channels and 8 attention heads that feeds into a 4-layer MLP segmentation head. It has 25,543,369 parameters and a size of 114.19 MB on disk. At predict time it uses 1.96GB of GPU RAM.

We compute loss pixelwise using Focal Loss [34] with a small downweighting of pixels towards the edge of the window. Our optimizer is AdamW [36], and we experiment with varying learning rate, weight decay, and perturb-scale (implementing the shrink perturb trick [1, 15]). We employ a OneCycle learning rate scheduler [52] with a cosine annealing strategy and starting fraction of 0.3. Our effective batch size is 24 with a real batch size of 2 and 12 accumulate gradient steps. This setup consumes approximately 20 GB of GPU RAM during training.

Prompt	Validation (n=691)				Test (n=121)			
	AP Box	AUC Box	F1 Box	TPR Box	AP Box	AUC Box	F1 Box	TPR Box
stool	0.01	0.03	0.07	0.07	0.05	0.08	0.18	0.13
droppings	0.02	0.10	0.14	0.23	0.08	0.14	0.27	0.30
petwaste	0.04	0.14	0.15	0.25	0.20	0.25	0.35	0.34
poop	0.04	0.10	0.17	0.16	0.17	0.18	0.31	0.26
dogpoop	0.05	0.16	0.17	0.20	0.24	0.28	0.38	0.39
caninefeces	0.05	0.16	0.18	0.29	0.17	0.24	0.37	0.39
turd	0.05	0.18	0.18	0.22	**0.27**	**0.32**	0.39	0.35
feces	0.06	0.21	0.18	0.27	0.16	0.26	0.32	0.39
excrement	0.07	**0.22**	0.20	0.28	0.25	0.31	0.39	**0.42**
dogfeces	0.07	0.21	**0.20**	**0.31**	0.23	0.29	**0.40**	0.38
animalfeces	**0.08**	0.21	0.20	0.25	0.23	0.30	0.39	0.38

Table 4. Zero-shot detection results with varied prompts. The chosen prompt has a significant impact on scores, and the best prompt is different between validation and test datasets, but overall zero-shot results are all low scoring. Because this is a zero-shot setting, the validation set can be compared to the test set. Interestingly, the validation scores significantly lower than the test scores indicating a greater degree of difficulty.

D.4.1. VIT-sseg Model Experiments

To establish a baseline, we evaluated 35 training runs where we varied input resolutions, window sizes, model depth, and other parameters. Although this initial search was somewhat ad-hoc, it provided insights into the optimal configuration for our model. Building on the best hyperparameters from this search, we performed a sweep over 7 combinations of learning rate, weight decay, and perturb scale (i.e., shrink and perturb [1, 15]). Scripts used to reproduce these experiments, as well as a log of the ad-hoc experiments, are available in the code repository. Additionally, trained models are packaged and distributed with information about their training configuration.

Note: the test dataset used in this appendix section is an older 30 image version with suffix `d8988f8c`, which is a subset of the more recent 121 image test set used in the main paper.

For each of the 7 hyperparameter combinations, we trained the model for 163,840 optimizer steps using a batch size of 24. We defined an “epoch” as 1,365 steps, at which point we saved a checkpoint, evaluated validation loss, and adjusted learning rates. To conserve disk space, we retained only the top 5 lowest-validation-loss checkpoints (although training crashes and restarts sometimes resulted in additional checkpoints, which are included in our evaluation).

Using the top-checkpoints, we predicted heatmaps for each image in the validation set. We then performed binary classification on each pixel (poop-vs-background) using a threshold. Next, we rasterized the truth polygons. The corresponding truth and predicted pixels were accumulated into a confusion matrix, allowing us to compute standard metrics such as precision, recall, false positive rate, etc. [45] for the specific threshold. By sweeping a range of thresholds, we calculated the average precision (AP) and the area under the ROC curve (AUC). We computed all metrics using scikit-learn [44]. Due to the high number of true negative pixels, we preferred AP as the primary measure of model quality.

The details of the top model for each run, along with relevant hyperparameters, are presented in Table 5. This table also includes the results on the small, held out, test set for the top model.

The results show strong performance on the validation set, with a maximum AP of 0.78. However, while the test AP for this model is good, it is significantly lower at 0.51. To investigate this discrepancy, we turned to qualitative analysis.

Qualitative results for the test, validation, and training sets are presented in Fig. 17. These examples illustrate both success and failure cases. The test and validation sets show clear responses to objects of interest, but the test set contains images of close-up and partially deteriorated poops. This suggests a bias in the dataset towards “fresh” poops taken from some distance.

Notably, the much larger training set also contains errors, indicating more information can be extracted from this dataset using hard-negative mining. There are clear difficult cases caused by sticks, leaves, pine cones, and dark areas on snow. We note that while compiling these results, we checked over 1000 images and discovered 14 cases where an object failed to be annotated, and it is likely that more are missed, but we believe these cases are rare.

Although focal loss was used, the current learning curriculum is likely under-weighting smaller distant objects. Our pixelwise evaluation metric is biased against this, which is a current limitation of our approach. Future work evaluating

Config Name	LR	Weight Decay	Perterb Scale	Validation (n=691) AP Pixel	Validation (n=691) AUC Pixel	Test (n=30) AP Pixel	Test (n=30) AUC Pixel
D05	1e-4	1e-6	3e-6	**0.7802**	**0.9943**	0.5051	0.9125
D03	1e-4	1e-5	3e-7	0.7758	0.9707	0.4346	0.8576
D04	1e-4	1e-7	3e-7	0.7725	0.9818	0.4652	0.7965
D02	1e-4	1e-6	3e-7	0.7621	0.9893	**0.5167**	**0.9252**
D00	3e-4	3e-6	9e-7	0.7571	0.9737	0.4210	0.7766
D01	1e-3	1e-5	3e-6	0.7070	0.9913	0.4607	0.9062
D06	1e-4	1e-6	3e-8	0.6800	0.9773	0.4137	0.8157

Table 5. Results for the best-performing models on the validation set across 7 hyperparameter configurations. The table provides detailed information about each configuration, including: 1) Configuration name (first column): a unique code identifying each training run used in the score scatter and box plots. 2) Varied hyperparameters (next three columns): specific values for learning rate, weight decay, and perturb scale that were used in each run. 3) Validation set performance (AP and AUC scores): metrics evaluating the model's performance on the validation set. 4) Test set performance (AP and AUC scores): metrics evaluating the model's performance on the test set using the same validation-maximizing models. Note that the top AP score over all models on the test set was 0.65, but it did not correspond to one of these validation runs used for model selection. Qualitative examples illustrating the performance of the top-scoring validation model listed here are provided in Fig. 17.

this dataset on an object-detection level can remedy this.

In Table 5 we only presented the top results. Here we've plotted the AP and AUC on the validation set for the top 5 AP-maximizing results from each of the 7 training runs. We also created a box-and-whisker plot for these top 5 results, which serves to assign a color and label to each training run. These plots are shown in Figure 18.

D.4.2. VIT Resource Usage

Note: we remind the reader that this section only applies to the VIT models.

All models were trained on a single machine with an 11900k CPU and a 3090 GPU. At predict time, using one background worker, our models processed 416×416 patches at a rate of 20.93Hz with 94% GPU utilization.

To better understand the energy requirements of our model, particularly for potential deployment on mobile devices, we used CodeCarbon [30] to measure the resource usage during prediction and evaluation. This analysis not only informs practical considerations but also helps us assess our contribution to the growing carbon footprint of AI [28]. The results for the 7 presented training experiments and the total 42 training experiments are reported in Table 6.

Direct measurement of resource usage during training is still under development, but we estimate the duration of each training run using indirect methods. We approximate energy consumption by assuming a constant power draw of 345W from the 3090 GPU during training. Emissions are estimated using a conversion ratio of 0.21 $\frac{\mathrm{kgCO_2}}{\mathrm{kWh}}$.

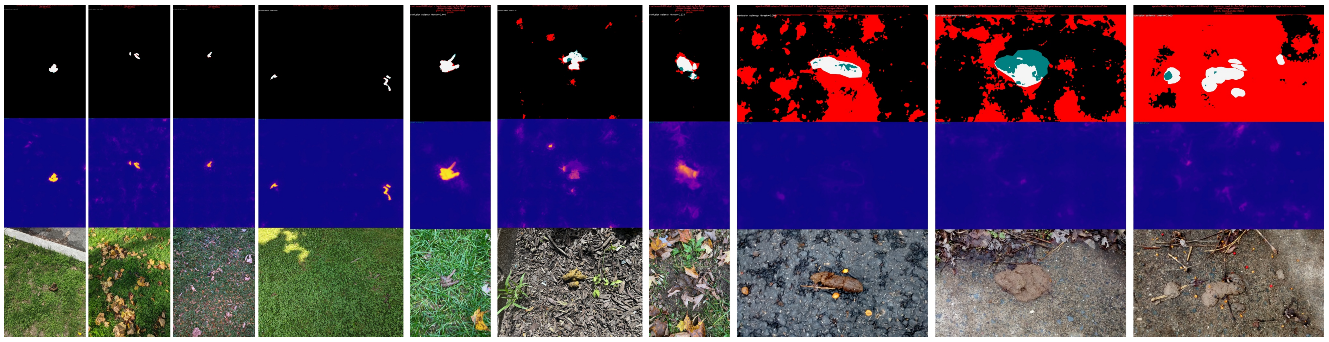
Based on the validation set's 691 images, we estimate that predicting on a single image on our desktop requires approximately 1.15 seconds and 0.13 Wh of energy. For context, typical mobile phones have a battery capacity of around 10 Wh and significantly less compute power than our desktop setup. While our models demonstrate the feasibility of training a strong detector from our dataset, they are not optimized for the mobile setting. To deploy our model on mobile devices, we will need to improve its efficiency or explore more efficient architectures.

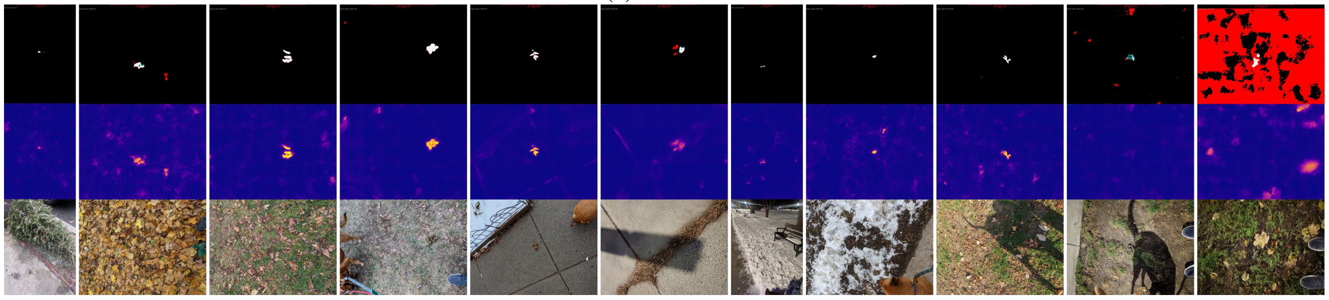
E. Environmental Impact

A footnote in the main paper reports the experiment costs, here we expand on the details. These costs are the total over all runs in the development of this paper over different datasets, with different numbers of runs per model, so it cannot be used to infer running time of the models. Instead it reports a component of the cost of performing this research. All costs are estimated assuming \$0.16 per kWh, \$25 per 1000 kg CO_2. The breakdown of resources used is given in Table 7.

Training accounted for the majority of resource usage with VIT models being the most expensive to run (Appendix D.4.2). The main reason is that VIT experiments operated on half-resolution images ($2{,}016 \times 1{,}512$) using 416×416 patches, whereas GroundingDINO, YOLO, and MaskRCNN were trained on smaller resized inputs (e.g., 640×640, 1066×800, depending on framework defaults) without windowing. A second reason is that we trained many more VIT variants in a hyperparameter search, which was done before easy to use foundational models became available. For training we estimated energy usage by measuring time and estimating GPU power draw approximated at 345W. To estimate emissions we used a factor of 0.21 kg CO_2/kWh.

For prediction resources estimates, there is an important limitation. The system running experiments was equipped


(a) Test set.


(b) Validation set.

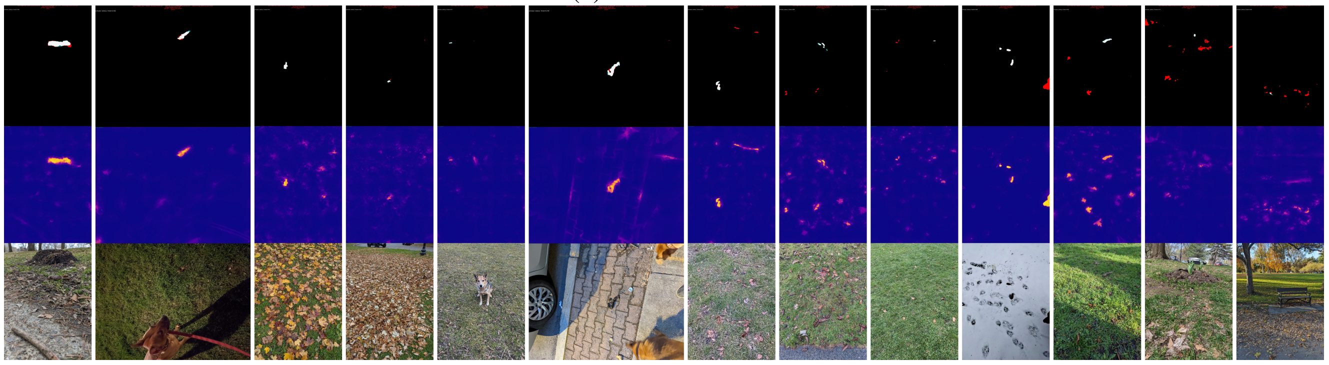

(c) Training set.

Figure 17. Qualitative results using the top-performing model on the validation set, applied to a selection of images from the (a) test, (b) validation, and (c) training sets. Success cases are presented on the left, with failure cases increasing towards the right. Each figure is organized into three rows: Top row: Binarized classification map, where true positive pixels are shown in white, false positives in red, false negatives in teal, and true negatives in black. The threshold for binarization was chosen to maximize the F1 score for each image, showcasing the best possible classification of the heatmap. Middle row: The predicted heatmap, illustrating the model's output before binarization. Bottom row: The input image, providing context for the prediction. The majority of images in the test set exhibit qualitatively good results. Failure cases tend to occur with close-up images of older, sometimes partially deteriorated poops. These examples were manually selected and ordered to demonstrate dataset diversity in addition to representative results.

with two RTX 3090 GPUs, although only a single one was used for any individual experiments. However, due to our use of CodeCarbon, which counts entire system resources, some double counting may have occurred when we ran two experiments simultaneously. Not all experiments were run in parallel, but some were. Still our estimates provide an upper bound for the resources utilization and we the lower bound will at best be half of our reported numbers. This limitation does not apply to our training estimations, which is the bulk of our cost, and thus our total numbers should only slightly inflated.

(a) Presented VIT experiment resources.

Node	Resource	Total	μ	n
eval	time	14.24 hours	0.41 hours	35
pred	time	11.97 hours	0.34 hours	35
pred	energy	8.76 kWh	0.25 kWh	35
pred	emissions	1.84 CO_2kg	0.05 CO_2kg	35
train*	time	39.22 days	5.60 days	7
train*	energy	324.75 kWh	46.39 kWh	7
train*	emissions	68.20 CO_2kg	9.74 CO_2kg	7

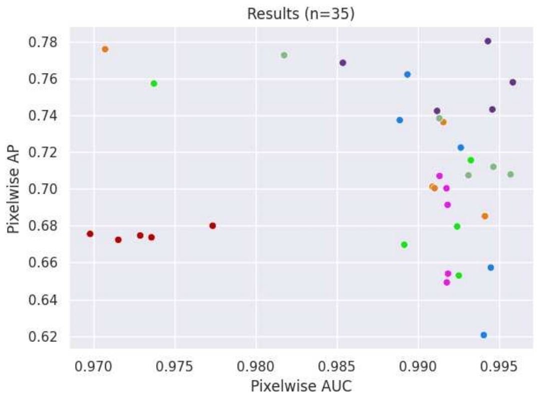
(b) All VIT experiment resources.

Node	Resource	Total	μ	n
eval	time	5.84 days	0.35 hours	399
pred	time	7.29 days	0.44 hours	399
pred	energy	102.83 kWh	0.26 kWh	399
pred	emissions	21.6 CO_2kg	0.05 CO_2kg	399
train*	time	158.95 days	3.78 days	42
train*	energy	1,316.07 kWh	31.34 kWh	42
train*	emissions	276.37 CO_2kg	6.58 CO_2kg	42

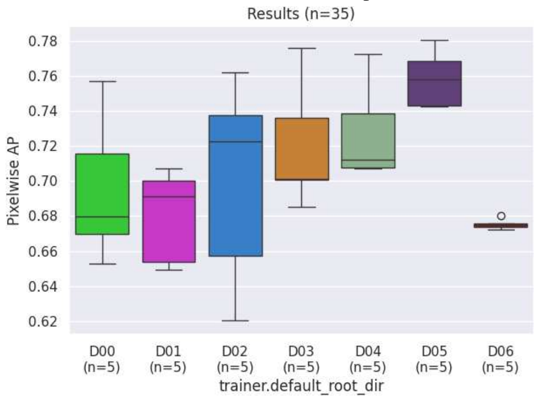
Table 6. Resources used for training, prediction, and evaluation. The "node" column is the pipeline stage: "train" for training, "pred" for heatmap prediction, and "eval" for pixelwise heatmap evaluation. The "resource" column lists the resource type: time, energy, or emissions. The "total" and "μ" columns show the total and average consumptions, and the "n" column indicates the frequency of each stage (e.g., across different hyperparameters). Train rows marked with an asterisk (*) are based on indirect measurements.

Phase	Model family	Time (days)	Energy (kWh)	Emissions (CO_2kg)	Cost (USD)
Train	VIT-sseg	158.95	1316.07	276.37	217.48
	MaskRCNN	0.71	5.92	1.24	0.98
	YOLO-v9	4.14	34.30	7.20	5.67
	Grounding DINO	0.32	2.68	0.56	0.44
	Total (training)	**164.12**	**1358.96**	**285.38**	**224.57**
Test	VIT-sseg	13.13	102.83	21.60	16.99
	MaskRCNN	0.57	4.41	0.93	0.73
	YOLO-v9	0.08	0.19	0.02	0.03
	Grounding DINO	0.13	0.29	0.02	0.05
	Total (prediction)	**13.91**	**107.72**	**22.57**	**17.80**
Overall total		**178.03**	**1466.69**	**307.95**	**242.37**

Table 7. Resource usage for training and prediction by model family. Time is wall-clock duration on a single RTX 3090. Energy is electricity consumed. Emissions use a 0.21 CO_2kg/kWh factor. Cost is estimated at $0.16/kWh electricity and $25 per 1000 CO_2kg.



(a) AP and AUC of 35 checkpoints.



(b) AP of 35 checkpoints.

Figure 18. (a) Scatterplot of pixelwise average precision (AP) and Area Under the ROC curve (AUC) for the top 5 checkpoints on the validation set. Points of the same color represent checkpoints from the same training run, which used identical hyperparameters. (b) Box-and-whisker plot the AP values across the top 5 checkpoints evaluated on the validation set. For each run, corresponding varied hyperparameters and maximum APs are given in Table 5.