From Instance Segmentation to Physical Quantification: A High-Resolution UAV Dataset for Façade Defect Assessment

Benyun Zhao[a], Jihan Zhang[a], Guidong Yang[a], Yijun Huang[a], Lei Lei[a], Xi Chen[a,*] and Ben M. Chen[a]

[a]*Department of Mechanical and Automation Engineering, The Chinese University of Hong Kong, Shatin District, N.T., Hong Kong*

ARTICLE INFO

Keywords:
Infrastructure Inspection
Building Inspection
Defect Detection
Unmanned Aerial Vehicle
Dataset

ABSTRACT

Visual inspection of civil infrastructure, particularly building façades, has long been an essential yet labor-intensive and high-risk task. Recent advances in unmanned aerial vehicle (UAV) systems and deep learning–based defect segmentation have substantially improved the efficiency and safety of visual inspection. However, despite significant progress in pixel-level defect localization, most existing approaches remain limited to visual detection, providing only 2D appearance cues without the metric or geometric information required for physical defect modeling. As a result, current segmentation outputs cannot reliably quantify severity indicators—such as crack width, crack propagation, or spalling volume—which fundamentally restricts their value for engineering assessment, maintenance prioritization, and automated decision-making. This limitation is amplified by the scarcity of high-quality datasets that not only offer pixel-level annotations but also support modeling defects in a physically meaningful, scalable manner. To address these gaps, this study introduces ***CUBIT-InSeg***, a high-resolution UAV-based façade defect dataset designed to advance the field from pixel-level segmentation toward physically grounded defect modeling. The dataset contains 6,996 high-definition images captured from diverse real-world building scenarios using a customised high-resolution UAV platform. CUBIT-InSeg focuses on two structurally critical defect types—cracks and spalling—chosen for their prevalence and strong relevance to severity assessment under engineering standards. Each image is annotated with precise instance-level masks to support geometric reconstruction and quantitative measurement. We conduct extensive benchmark evaluations on more than 18 state-of-the-art segmentation models, providing a comprehensive performance analysis and establishing a strong baseline for subsequent modeling tasks. Furthermore, zero-shot deployments on real-world building façades demonstrate the practical robustness and applicability of models trained on CUBIT-InSeg. By bridging the gap between visual segmentation and physical defect modeling, this work provides a foundational dataset and benchmark that pave the way for scalable, autonomous, and quantitatively informed façade defect assessment.

1. Introduction

Civil infrastructure is vulnerable to damage caused by a multitude of factors such as weather impacts, external loads, structural deterioration, and poor design. Periodic infrastructure inspections are crucial for remaining safe and functional infrastructures. Currently, non-destructive testing (NDT) devices like optical cameras [] [1], laser scanners [2], impact echo [3], and ground-penetrating radar [4] are used for manual defect detections in civil infrastructure. Although human visual inspection is the most flexible and feasible method for preliminary diagnosis, it is subjective, time-consuming, laborious, and error-prone. It can also pose significant health and safety risks to human inspectors, especially when inspecting high-rise buildings and large spaces. To overcome these challenges, robotic platforms like unmanned aerial vehicles (UAVs) and unmanned ground vehicles (UGVs) [5,6] have been developed to achieve more accurate and efficient infrastructure inspections, from data collection and defect analysis. These unmanned platforms integrating computer vision techniques help achieve better inspection results.

In recent years, automatic image processing technologies driven by deep learning methods [7–10] have achieved remarkable breakthroughs, demonstrating substantial advantages in both efficiency and effectiveness compared with traditional image processing techniques [11,12]. Among these, instance segmentation—a task requiring pixel-level understanding—plays a pivotal role not only in achieving precise defect localization but also in enabling quantitative defect analysis, thereby showcasing the superior capability of deep learning in tackling complex visual challenges. Consequently, an increasing number of researchers in the architecture, engineering, and construction (AEC) domain [13,14] have shifted toward deep learning–based segmentation approaches for the inspection and management of infrastructure defects.

However, deep learning algorithms are notoriously data-hungry, demanding large, high-quality, and domain-specific datasets tailored to the characteristics of building defect segmentation tasks. Most existing segmentation models are trained on general-purpose open-source datasets such as MS COCO [17], which feature abundant images, diverse object categories, and complex scene compositions. In contrast, the collection and pixel-level annotation of defect-related images in civil infrastructure pose unique challenges due

xichen002@cuhk.edu.hk (Xi Chen)

[1]CUBIT stand for CUHK Building Information Technology.

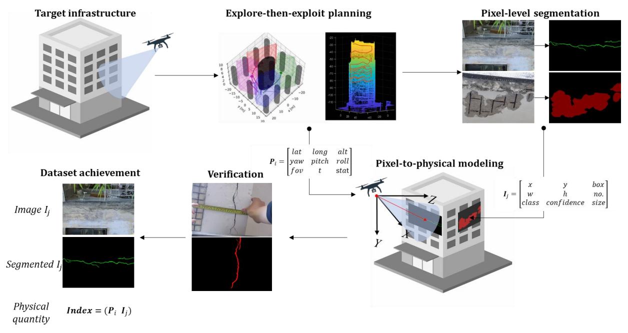

Target infrastructure
Explore-then-exploit planning
Pixel-level segmentation
$P_i = \begin{bmatrix} lat & long & alt \\ yaw & pitch & roll \\ fov & t & stat \end{bmatrix}$
Pixel-to-physical modeling
$I_j = \begin{bmatrix} x & y & box \\ w & h & no. \\ class & confidence & size \end{bmatrix}$
Dataset achievement
Verification
Image I_j
Segmented I_j
Physical quantity
$Index = (P_i \ I_j)$
Z
X
Y

Figure 1: The overall pipeline of the our pixel-to-physical modeling for infrastructure defects.

to the complexity and dynamic nature of construction environments. As a result, there remains a significant scarcity of well-curated, instance-level segmentation datasets specifically designed for the building and infrastructure sector.

Despite the rapid progress of AI-driven visual inspection for civil infrastructure, existing UAV- or camera-based approaches largely focus on detecting or segmenting surface-level defects without providing the geometric or physical attributes required for engineering-grade assessment. As highlighted by recent studies, the community has reached a turning point where defect modeling—rather than mere defect detection—is becoming essential for structural health monitoring [1]. Purely image-based recognition, even at pixel-level precision, cannot convey crucial physical information such as defect dimensions, depth, or volumetric loss, which are fundamental for quantifying severity, prioritising maintenance, and complying with engineering standards. Consequently, systems relying solely on 2D visual cues struggle to support downstream decision-making and cannot provide inspectors with actionable parameters linked to structural integrity or deterioration mechanisms.

To address this gap, a growing body of research has begun exploring more sophisticated forms of defect modeling. For instance, pixel-level reconstruction combined with photogrammetric texture mapping has been used to derive 3D crack representations [2]. Volumetric assessment using depth-enhanced imaging further illustrates the potential of geometric cues for evaluating damage extent [3]. More recently, methods leveraging 3D point clouds and dynamic graph convolutional networks have achieved promising results in constructing as-inspected defect models that capture both geometric structure and semantic attributes [4]. Alongside these developments, advanced segmentation approaches—including SAM-based models [5] and weakly supervised or scribble-annotation-based pipelines [6]—demonstrate notable progress toward generalisable and annotation-efficient defect understanding. Furthermore, deep learning frameworks integrating appearance, texture, and material characteristics show potential for more accurate defect quality assessment [7].

However, a persistent limitation remains: most publicly available datasets are exclusively image-based and lack the physical metrics necessary for accurate defect modeling. Whether designed for detection, segmentation, or instance-level labeling, existing datasets typically provide only RGB visual information without corresponding 3D geometry, calibrated physical scales, or standardized severity labels. This absence of physically grounded information hampers real-world deployment in several ways: (i) defect severity cannot be quantified due to the absence of metric scale; (ii) algorithmic generalization is limited, particularly when transferring from curated datasets to large-scale façade environments; and (iii) autonomous UAV operation becomes difficult, as the lack of metric cues impedes planning, standoff control, and automated follow-up inspection.

These limitations underscore the urgent need for high-resolution, physically meaningful, UAV-derived defect datasets that support not only visual recognition but also the modeling, measurement, and interpretation of defects within the context of engineering standards. This motivation directly leads to the key contributions of our work: (i) a UAV-acquired, high-resolution façade defect dataset that includes both common and severe defects—not limited to cracks but also covering spalling, which is emphasized in ISO-based severity indicators; (ii) extensive benchmarks including two cross-domain datasets to evaluate robustness and transferability; and (iii) a quantitative zero-shot evaluation on real-world scenes demonstrating the deployability of the proposed system in autonomous inspection scenarios.

2. Related Work

With the rapid advance of UAV-based inspection platforms and deep learning techniques, research in civil infrastructure monitoring has moved beyond basic defect detection toward a more ambitious goal: modeling defects in a physically meaningful manner. While earlier systems mainly focused on identifying cracks or spalling regions from images [8, 9], an increasing body of work now emphasises the need to extract geometric, metric, and structural attributes of defects rather than treating them solely as 2D visual patterns.

This shift is reflected in several recent research directions. One prominent line of work integrates multi-view photogrammetry or depth-enhanced imaging to reconstruct cracks or damaged areas in three-dimensional space, enabling volumetric or shape-based assessment [2, 3]. Similarly, studies leveraging 3D point clouds and graph-based semantic segmentation have demonstrated the feasibility of building as-inspected defect models that encode both geometry and material characteristics of concrete surfaces [4]. Beyond purely image-driven approaches, researchers have also explored aligning UAV imagery with Building Information Models (BIM) to achieve defect-level reconstruction within semantically rich building geometry [10]. Recent developments further illustrate the integration of UAV sensing, AI, and GeoBIM into high-precision digital twin frameworks that directly embed defect information into geometric and

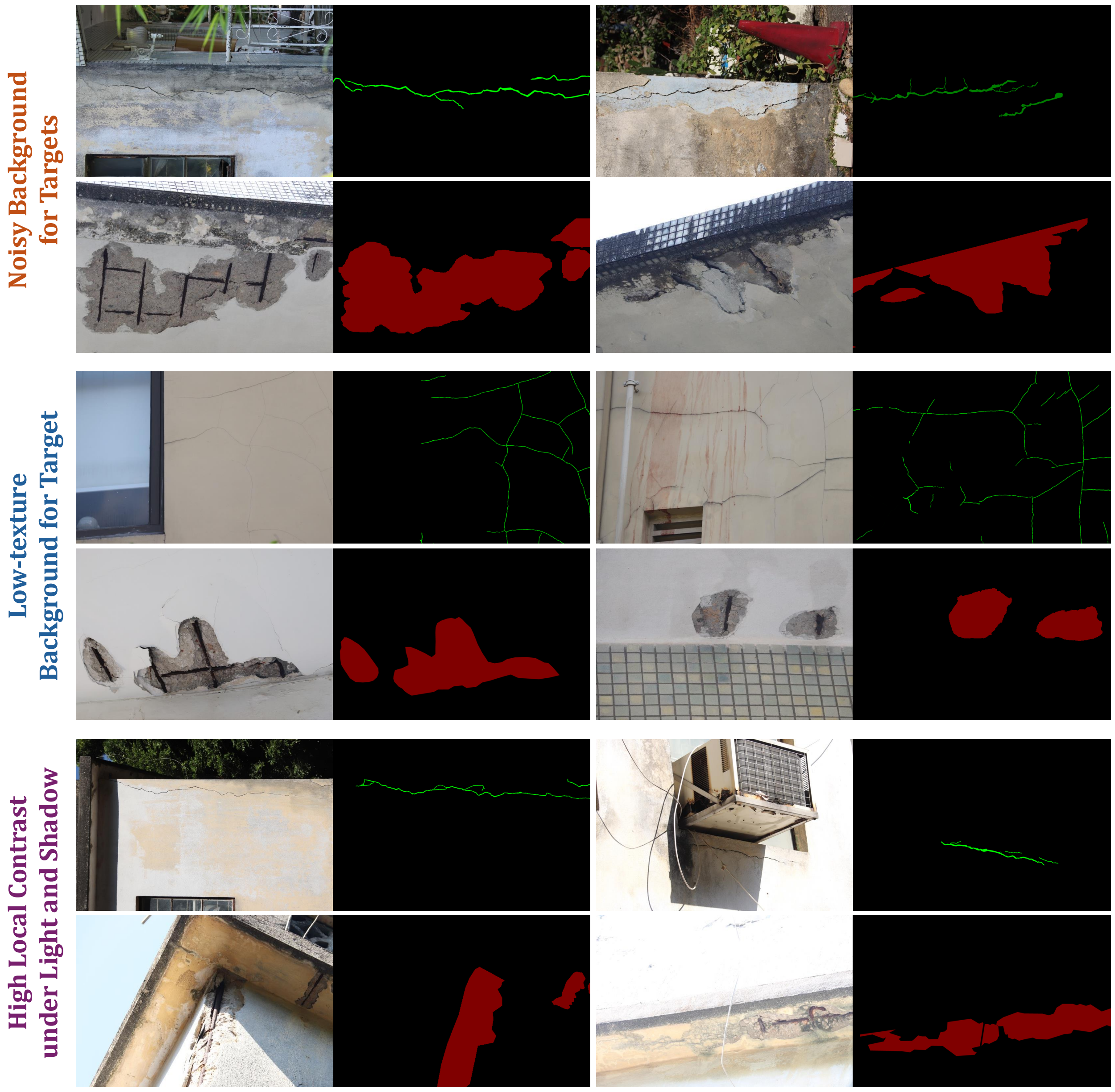

Noisy Background for Targets
Low-texture Background for Target
High Local Contrast under Light and Shadow

Figure 2: Sample of our proposed ***CUBIT-InSeg*** dataset.

lifecycle management systems [11]. Collectively, these studies highlight a clear trajectory: the field is transitioning from what a defect looks like to what it physically means for structural safety and maintenance planning.

However, despite rapidly growing interest in defect modeling, the majority of existing datasets remain limited to image-only crack or spalling annotations—typically at the bounding GT-Box or pixel level—without providing the geometric, metric, or severity-related information necessary for physically grounded modeling. As recent reviews point out [1], the absence of datasets capable of supporting defect geometry reconstruction or physical-scale estimation has become a key barrier for the development and evaluation of modeling-oriented algorithms. Moreover, the lack of high-resolution UAV datasets capturing diverse façade defects under real operational conditions further restricts the robustness and generalization of such methods.

2.1. Comparison with Existing Infrastructure Defect Segmentation Datasets

To highlight the characteristics of the proposed CUBIT-InSeg dataset, we compare it with existing infrastructure defect segmentation datasets in Table 1. Most publicly available datasets primarily target road and pavement scenarios, resulting in limited diversity in defect types, imaging perspectives, and environmental conditions. Road-focused datasets such as GAPs384 [12], EdmCrack600 [13], and GAPs-10m [14] provide pixel-level annotations but are restricted to ground-level imaging. Highway-Crack [15] is the

Table 1
The Comparison between Other Unmanned System-captured Defects Segmentation Dataset with our *CUBIT-InSeg*

Dataset	Image Volume	Resolution	Data Collection Platform	Defect Type	Infrastructure	Task Type
GAPs384 [12]	384	1920×1080	Ground Vehicle	Crack	Pavement	Pixel Level
GAPs-10m [14]	20	5030×11505	Ground Vehicle	Crack	Pavement	Pixel Level
EdmCrack600 [13]	600	1920×1080	Ground Vehicle	Crack	Pavement	Pixel Level
Highway-Crack [15]	4,118	512×512	Unmanned Aerial Vehicle	Crack	Highway	Pixel Level
Crack-Seg [16]	4,029	416×416	Ground Vehicle	Crack	Building Pavement	Pixel Level
UAV75 [17]	75	512×512	Unmanned Aerial Vehicle	Crack	Building	Pixel Level
CUBIT-InSeg (*Ours*)	**6,996 62,178 instances**	$\mathbf{4800 \times 3200}$	**Unmanned Aerial Vehicle**	**Crack Spalling**	**Building**	**Instance Level**

only UAV-based dataset in this group, containing 4,118 post-earthquake highway images; however, it remains constrained to roadway surfaces.

For building facade defects, publicly available datasets are extremely limited. Crack-Seg [16] includes 4,029 images covering pavements and partial building scenes, yet lacks true aerial viewpoints and the high resolution required for UAV inspection scenarios. UAV75 [17] provides UAV imagery but contains only 75 low-resolution samples, making it unsuitable for training modern deep segmentation models. Moreover, existing datasets overwhelmingly focus solely on cracks and do not include more severe facade defects such as spalling, which are critical for structural safety. According to established building pathology standards[1] and professional inspection guidelines issued by the Hong Kong Buildings Department[2] and the Hong Kong Institute of Surveyors, spalling of the concrete cover is classified as a high-risk defect: it is commonly induced by prolonged moisture exposure, oxidation of reinforcement, and the progressive widening of pre-existing cracks, and may lead to detachment of concrete or even localized collapse if left unattended. Guided by these standards and by our prior defect taxonomy studies, CUBIT-InSeg explicitly includes spalling—along with cracks—to better reflect the defect types of greatest concern in real-world facade safety assessment.

Another limitation observed across these datasets is their generally low image resolution and low defect density: most images contain only a single defect instance, which does not reflect real-world UAV inspection conditions where multiple, spatially distributed defects commonly appear within the same facade view. Our CUBIT-InSeg addresses the aforementioned limitations by providing high-resolution UAV imagery, diverse defect types (crack and spalling), complex multi-instance scenarios, and realistic aerial inspection perspectives, thereby offering a comprehensive and practically relevant benchmark for building defect segmentation.

2.2. Path Planning for Data Collection

In this study, we develop an equal-distance UAV imaging framework that enables high-quality data acquisition for digital-twin-based façade defect modeling. The key objective is to ensure that each image is captured from a prescribed, nearly constant standoff distance to the building surface, so that pixel-level defect segmentation can be consistently linked to physical dimensions (e.g., crack width, spalling area and depth) in the reconstructed 3D model. Our framework extends recent explore–then–exploit multi-UAV coverage schemes for infrastructure inspection and reconstruction [18, 19], and integrates them with depth-aware surface modeling, equal-distance viewpoint generation, and DT updating for defect-aware asset management.

We adopt an explore–then–exploit paradigm similar to recent hierarchical multi-UAV frameworks for building inspection. During the exploration stage, each UAV performs depth-visual SLAM to estimate its six-degree-of-freedom pose and incrementally reconstruct a dense point cloud of the façade and nearby obstacles. The environment is discretised into a 3D occupancy grid, where each voxel stores (i) occupancy probability, (ii) distance to the nearest obstacle, and (iii) a reconstructability score.

The reconstructability is computed by combining stereo-geometry and light-field principles: voxels that can be observed from multiple viewpoints with favourable parallax angles and sufficient field-of-view coverage receive higher scores, while voxels that are too close to obstacles or outside the effective sensing range are penalised. This results in a density map $R(\mathbf{p})$ that reflects both the reconstructability of the façade and safety margins around obstacles.

Let $W \subset \mathbb{R}^3$ denote the workspace, B the building volume, and $\{\mathbf{s}_k\}$ the set of surface points of the façade estimated from the depth-based SLAM. The density map is initialised using a coarse bounding box of B and is iteratively updated as more points are observed.

To coordinate multiple UAVs, we employ a Voronoi-based spatial deployment strategy inspired by distributed coverage control. Let $P(t) = [\mathbf{p}_1(t), \ldots, \mathbf{p}_n(t)]$ be the UAV positions at time t. The workspace W is partitioned into non-overlapping Voronoi cells $V_i(t)$ such that each UAV i is responsible for coverage within its own cell. The density-weighted coverage cost

$$H(P) = \sum_{i=1}^{n} \int_{V_i} \|\mathbf{p} - \mathbf{p}_i\|^2 R(\mathbf{p})\, d\mathbf{p} \qquad (1)$$

[1] British Standards Institution standards publication about "building and constructed assets – service life planning" BS ISO 15686-7:2017

[2] Volume 1: Pre-1980 Residential&Composite Buildings in Hong Kong

is minimised by a gradient-descent control law that drives the UAVs towards a centroidal Voronoi tessellation. This yields a load-balanced spatial deployment where each UAV converges to the “best” region of the façade in terms of reconstructability and safety.

To guarantee collision avoidance, we follow the hyperplane-based construction of safe convex corridors between UAVs and obstacles. The Voronoi cells are intersected with these corridors to ensure that the motion of each UAV remains within a collision-free region while the global coverage cost is reduced.

Once the exploration stage converges and a sufficiently accurate façade surface model is obtained, we switch to the exploitation stage to generate equal-distance viewpoints for high-quality imaging and defect modeling. For each surface point $\mathbf{s}_k$ with outward normal $\mathbf{n}_k$, we define a desired camera position

$$\mathbf{v}_k = \mathbf{s}_k + d_{\text{target}}\, \mathbf{n}_k, \tag{2}$$

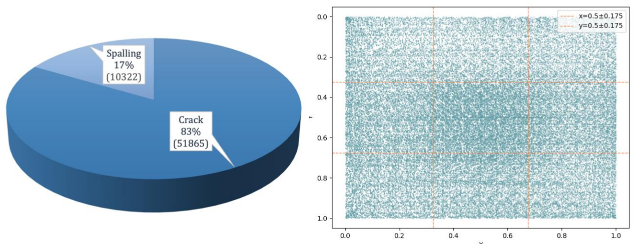
where d_{target} is the prescribed standoff distance, chosen based on camera field-of-view, required ground-sample distance (GSD) for defect segmentation, and safety requirements.

To avoid redundancy, the façade is first discretised into small patches (e.g., in the (u, v) parameter space of the surface), and one or several viewpoints are generated per patch such that: (i) the angle between the viewing direction and the surface normal is within a specified bound, (ii) the overlap between neighbouring images exceeds a minimum threshold, and (iii) the viewpoints lie within the collision-free corridors and do not violate minimum distance-to-obstacle constraints. This procedure yields a set of candidate equal-distance viewpoints $\mathcal{V} = \{\mathbf{v}_1, \ldots, \mathbf{v}_{N_v}\}$ for each UAV.

After viewpoint generation, the viewpoints assigned to each UAV are further partitioned into capacity-constrained subregions by applying a capacity-constrained Voronoi tessellation inside its working cell. The capacity of each subregion is defined in terms of (i) the number of viewpoints and (ii) the estimated travel cost, which reflects the limited endurance of each UAV.

For each subregion, we formulate a trajectory-based travelling salesman problem (TSP): the edges between viewpoints are weighted by collision-free path lengths obtained with an A* or kinodynamic planner in the occupancy map. Solving the TSP for each subregion yields a set of short, feasible routes that visit all viewpoints while minimising inspection time.

The resulting routes are then converted into an automatic flight plan by exporting time-parameterised waypoints (position, yaw, and desired camera trigger events) in the format required by the UAV autopilot (e.g., MAVLink mission file). In this way, the proposed methodology extends existing coverage-control algorithms from “high-level exploration” to a fully integrated, DT-ready automatic route planner for equal-distance façade imaging.



(a) Proportion of Defect Categories (b) Scatter Plot of Object Position Distribution

Figure 3: (a) Distribution of annotated bounding box sizes for defects, (b) Distribution of sizes for sampled non-overlapping background bounding boxes.

2.3. Statistics and Target Position Distribution

Figure 3 presents the statistical summary and spatial distribution of defect instances in the CUBIT-InSeg dataset. Among the 6,996 images, a total of 62,187 annotated targets are included, of which *crack* accounts for 51,865 instances (83%) and *spalling* for 10,322 instances (17%), as shown in Fig. 3(b). This imbalance highlights the natural rarity of spalling in real infrastructure scenes. Although less frequently observed, spalling represents a more severe façade defect with higher safety risks, underscoring the need for increased attention and dedicated data collection despite its lower occurrence.

Beyond instance counts, the spatial distribution of targets is another critical factor in evaluating an instance segmentation dataset. The placement of defects within images influences a model’s *spatial awareness*, which represents its ability to recognize objects across different locations, and directly affects robustness to target offset or positional shifts. In practical UAV inspection scenarios, defects often appear sparsely and unpredictably across the façade. Models trained on datasets with well-distributed targets are therefore more capable of detecting defects located near image borders, in corners, or in unconventional positions. Figure 3(a) visualizes the scatter plot of ground-truth bounding-box centers for all defect instances. The distribution is reasonably uniform across the image plane, with a moderate concentration near the central region: approximately 63% of all targets (39,078 instances) fall within the normalized range $[x, y] \in \{0.5{\pm}0.175\}$. Such a distribution ensures that models trained on our CUBIT-InSeg dataset develop strong spatial generalization capabilities and exhibit higher robustness when deployed in real-world UAV inspection scenarios.

2.4. Foreground and Background Box Sizes

To further characterize the geometric variability of CUBIT-InSeg, Fig. 4(a) illustrates the size distribution of ground-truth defect bounding boxes. The defects span an exceptionally wide scale range: crack instances form thin, elongated structures with short sides sometimes below 5 pixels, whereas spalling regions may exceed 4800 pixels along their longer dimension. The strong concentration of points near the lower boundary of the "Smaller side" axis reflects the high anisotropy and extreme slenderness of cracks, while the marginal histograms reveal a long-tailed

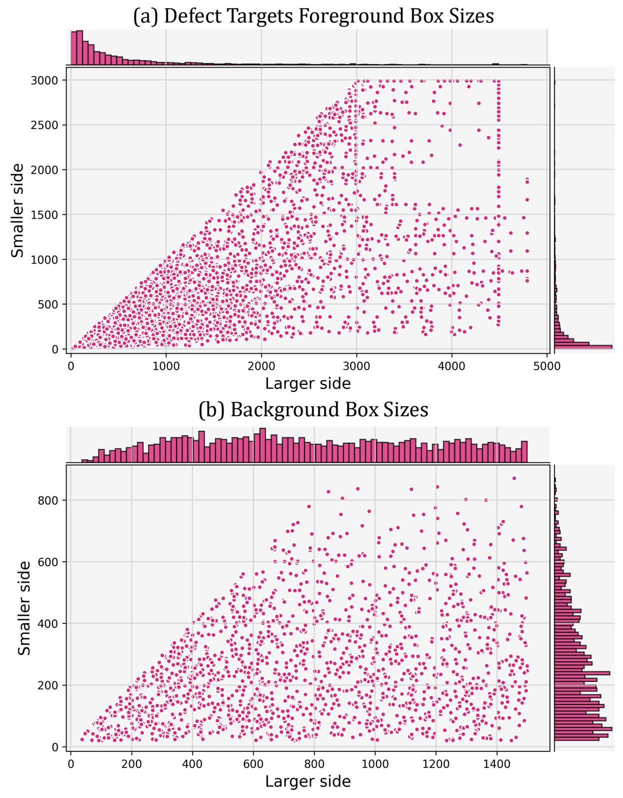


Figure 4: (a) Distribution of annotated bounding box sizes for defects, (b) Distribution of sizes for sampled non-overlapping background bounding boxes.

distribution covering both microscopic and very large facade defects. This pronounced scale diversity poses a significant challenge for both detection and segmentation.

Fig. 4(b) shows the distribution of background boxes, which are randomly sampled non-defect patches used as a reference for contrastive analysis. These background patches are intentionally constrained to moderate sizes so that they remain representative of local non-defect regions. The empty region in the lower-left corner is expected: background boxes smaller than 20 pixels on either side are excluded, corresponding to the 20-pixel receptive field induced by the 32× downsampling operation (P5-stage) in common instance segmentation models (input size 640 / 32 = 20). Patches below this scale do not constitute meaningful semantic context for the model and are therefore omitted. Compared with the highly heterogeneous foreground distribution, the background boxes exhibit a more compact and uniform pattern, highlighting the intrinsic difficulty of detecting tiny cracks that may occupy less than 0.01% of the image while simultaneously handling large-scale structural defects. Together, these characteristics underscore the geometric richness and real-world complexity embodied in the CUBIT-InSeg dataset.

2.5. Physically Based Defect Measurement from Pixel-Level Segmentations

Given the equal-distance UAV imaging and automatic flight planning strategy described in the previous subsection, all façade images in CUBIT-InSeg are captured at a prescribed, nearly constant standoff distance d_{target} from the building surface. This imaging geometry enables pixel-level defect segmentations to be directly converted into physically meaningful measurements, which are later used for DT-based defect quantification with the workflow in Fig. 5.

Let I_i denote the i-th UAV image captured at distance d_{target} with camera focal length f and pixel pitch δ_p (physical size of one pixel on the sensor). Under the pinhole camera model, the ground-sample distance (GSD) on the façade plane can be approximated as

$$\text{GSD} = \frac{d_{\text{target}}}{f}\,\delta_p, \tag{3}$$

which, thanks to the equal-distance imaging strategy, is assumed to be approximately constant across all images and within each façade patch. Consequently, one pixel corresponds to a fixed physical length GSD on the façade, and an axis-aligned pixel square relates to a physical area of GSD^2.

Using the instance-level segmentation model trained on CUBIT-InSeg, each image I_i is associated with a set of defect instances

$$\mathcal{M}_i = \{\Omega_i^{(k)}\}_{k=1}^{N_i},$$

where $\Omega_i^{(k)} \subset \mathbb{Z}^2$ denotes the pixel set of the k-th defect instance (either crack or spalling) in image I_i.

For crack-type instances, we first compute a skeletonised representation to decouple crack *length* and *width*. Let $\Gamma_i^{(k)}$ be the skeleton pixels of instance $\Omega_i^{(k)}$, extracted by a standard thinning algorithm, and let $\mathbf{t}(\mathbf{p})$ be the unit tangent direction along the skeleton at pixel $\mathbf{p} \in \Gamma_i^{(k)}$. The corresponding normal direction is

$$\mathbf{n}_\perp(\mathbf{p}) = \mathbf{R}_{90^\circ}\,\mathbf{t}(\mathbf{p}),$$

where $\mathbf{R}_{90^\circ}$ rotates a 2D vector by 90°.

Crack width. For each skeleton pixel $\mathbf{p}$, we count the number of consecutive crack pixels along $\mathbf{n}_\perp(\mathbf{p})$ that remain inside $\Omega_i^{(k)}$:

$$N_\perp(\mathbf{p}) = \left|\{\mathbf{q} \in \Omega_i^{(k)} \mid \mathbf{q} \text{ lies on the normal line through } \mathbf{p}\}\right|.$$

The local physical crack width at $\mathbf{p}$ is then

$$w_i^{(k)}(\mathbf{p}) = N_\perp(\mathbf{p})\,\text{GSD}. \tag{4}$$

An instance-level crack width can be defined as the mean or maximum of $w_i^{(k)}(\mathbf{p})$ over all $\mathbf{p} \in \Gamma_i^{(k)}$, e.g.

$$\bar{w}_i^{(k)} = \frac{1}{|\Gamma_i^{(k)}|}\sum_{\mathbf{p}\in\Gamma_i^{(k)}} w_i^{(k)}(\mathbf{p}). \tag{5}$$

Crack length. Similarly, the crack length is obtained by summing the physical distances between adjacent skeleton pixels along $\Gamma_i^{(k)}$. Let $\Gamma_i^{(k)} = \{\mathbf{p}_1, \ldots, \mathbf{p}_{L_k}\}$ be ordered along the crack centreline; the physical length is approximated as

$$L_i^{(k)} \approx \sum_{j=1}^{L_k-1} \|\mathbf{p}_{j+1} - \mathbf{p}_j\|_2 \, \mathrm{GSD}. \tag{6}$$

For spalling-type instances, the primary geometric descriptor at the image level is the defect area. Given an instance mask $\Omega_i^{(k)}$ labelled as spalling, its area in pixels is simply $|\Omega_i^{(k)}|$; the corresponding physical area is

$$A_i^{(k)} = |\Omega_i^{(k)}| \, \mathrm{GSD}^2. \tag{7}$$

Additional shape descriptors, such as equivalent diameter, aspect ratio, or compactness, can be derived from the pixel mask and converted to physical units by scaling lengths with GSD. These descriptors form a physically consistent feature set for spalling, which is particularly important given its higher severity in façade safety assessment compared with cracks.

The above procedure yields, for each defect instance $\Omega_i^{(k)}$, a collection of physically interpretable measurements, e.g.,

$$\left(L_i^{(k)}, \bar{w}_i^{(k)}, A_i^{(k)}, \text{shape features, defect type}\right),$$

all expressed in metric units under the equal-distance imaging assumption. These image-level measurements are aggregated at façade- or component-level (e.g., by grouping instances within the same façade panel or elevation zone), providing a compact statistical description of defect conditions.

2.6. Physically Based Defect Quantification on the Digital Twin

Building upon the physically measurable crack and spalling descriptors derived from pixel-level segmentations, this section integrates these measurements into a DT representation of the façade. The goal is to transform 2D image–based defect indicators into component-level condition assessments that align with international building pathology standards, including BS ISO 15686-7:2017 and the inspection guidelines issued by the Hong Kong Buildings Department and the Hong Kong Institute of Surveyors.

Let the façade DT be composed of M surface or BIM elements $\{\mathcal{E}_j\}_{j=1}^{M}$. Each defect instance detected in images is associated with one or more façade elements based on its image footprint and field-of-view projection.

For each defect instance k in image I_i, we have the set of physical measurements:

$$\Phi_i^{(k)} = \{L_i^{(k)},\ \bar{w}_i^{(k)},\ A_i^{(k)},\ \text{shape},\ \text{type}\}.$$

We assign these measurements to façade element $\mathcal{E}_j$ if the defect appears within the region of the façade that is imaged by the camera when capturing I_i. Let $\mathcal{K}_j$ denote the set of all defect instances assigned to $\mathcal{E}_j$.

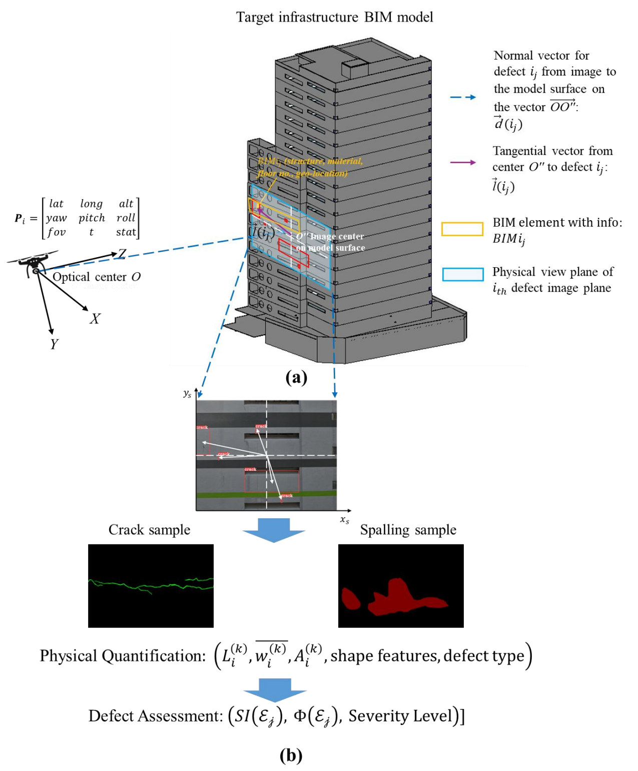

Target infrastructure BIM model
$\boldsymbol{P}_i = \begin{bmatrix} lat & long & alt \\ yaw & pitch & roll \\ fov & t & stat \end{bmatrix}$
Z
Optical center O
X
Y
$BIMi_j$ (structure, material, floor no., geo-location)
$\vec{l}(i_j)$
O'' Image center on model surface
Normal vector for defect i_j from image to the model surface on the vector $\overrightarrow{OO''}$: $\vec{d}(i_j)$
Tangential vector from center O'' to defect i_j: $\vec{l}(i_j)$
BIM element with info: $BIMi_j$
Physical view plane of i_{th} defect image plane
(a)
y_s
x_s
crack
Crack sample
Spalling sample
Physical Quantification: $\left(L_i^{(k)}, \overline{w_i^{(k)}}, A_i^{(k)}, \text{shape features, defect type}\right)$
Defect Assessment: $(SI(\mathcal{E}_j),\ \Phi(\mathcal{E}_j),\ \text{Severity Level})]$
(b)

Figure 5: Physically based façade defect quantification workflow: (a) defect registration; (b) defect assessment.

The aggregated defect state of element $\mathcal{E}_j$ is represented as

$$\Phi(\mathcal{E}_j) = \bigcup_{k \in \mathcal{K}_j} \Phi_i^{(k)}.$$

For practical analysis, we compute façade–element–level statistics:

$$\mathrm{CrackLength}(\mathcal{E}_j) = \sum_{k \in \mathcal{K}_j,\ \text{type}=C} L_i^{(k)}, \tag{8}$$

$$\mathrm{MeanCrackWidth}(\mathcal{E}_j) = \frac{1}{|\mathcal{K}_j^C|} \sum_{k \in \mathcal{K}_j^C} \bar{w}_i^{(k)}, \tag{9}$$

$$\mathrm{SpallingArea}(\mathcal{E}_j) = \sum_{k \in \mathcal{K}_j,\ \text{type}=S} A_i^{(k)}, \tag{10}$$

where $\mathcal{K}_j^C$ and $\mathcal{K}_j^S$ denote the crack and spalling instances, respectively.

These metrics constitute the defect signature of each façade component and serve as the input for severity grading and maintenance prioritization.

According to BS ISO 15686-7:2017 [20], the Hong Kong Buildings Department (BD) [21], and the Hong Kong Institute of Surveyors (HKIS) [22], two defects are regarded as highly safety-critical:

Table 2
Severity classification for façade defects based on physical measurements and ISO/HK practice

Level	Description	Typical Thresholds	Action
Low (SI < 0.25)	Minor deterioration	Crack width < 0.2 mm No spalling	Routine monitoring
Moderate ($0.25 \leq$ SI < 0.50)	Non-structural impact	0.2–0.5 mm cracks Small spalling < 50 cm^2	Repair scheduling
Severe ($0.50 \leq$ SI < 0.75)	Significant safety concern	Cracks > 0.5 mm Spalling 50–200 cm^2	Urgent repair
Critical (SI ≥ 0.75)	High risk of failure	Wide cracks > 1 mm Large spalling > 200 cm^2	Immediate action / cordon-off

- **Cracks**: Risk of moisture ingress, reinforcement corrosion, propagation into spalling.
- **Spalling**: Classified as a high-risk defect that may lead to detachment of concrete cover and localized collapse.

To align with these standards, we define a unified **Severity Index (SI)** for each façade element:

$$SI(\mathcal{E}_j) = \alpha_1 W_C(\mathcal{E}_j) + \alpha_2 L_C(\mathcal{E}_j) + \beta_1 A_S(\mathcal{E}_j),$$

where:
- $W_C(\mathcal{E}_j)$ = normalized mean crack width - $L_C(\mathcal{E}_j)$ = normalized total crack length - $A_S(\mathcal{E}_j)$ = normalized total spalling area - $\alpha_1, \alpha_2, \beta_1$ = weights reflecting relative safety impact (typically $\beta_1 > \alpha_1 > \alpha_2$ due to the high risk of spalling)

Normalization is performed with respect to ISO guideline thresholds and BD practice notes.

Using the Severity Index, we define four façade defect levels consistent with ISO 15686 and Hong Kong inspection practice.

These thresholds reflect:

- ISO 15686's durability and condition-rating guidance
- Hong Kong BD's classification of "defective concrete cover"
- HKIS's façade safety inspection criteria

The final DT representation stores the severity level, numerical SI value, and detailed defect metrics for each façade component. This enables automatic maintenance scheduling, lifecycle cost estimation, and longitudinal tracking of defect evolution within the DT environment.

Each façade element $\mathcal{E}_j$ in the DT is annotated with:

$$\big(SI(\mathcal{E}_j),\ \Phi(\mathcal{E}_j),\ \text{Severity Level}\big),$$

allowing users to:

- visualize defect locations and severities on the DT model;
- perform time–series monitoring as new UAV inspections are collected;
- support automated condition assessment reports;
- prioritize repairs based on quantitative risk levels.

This establishes a full pipeline from UAV image acquisition and pixel-level segmentation to physically grounded DT-based structural assessment.

3. Benchmark Experiments of the Proposed CUBIT-InSeg Dataset

To comprehensively evaluate the proposed CUBIT-InSeg dataset, we trained an extensive suite of deep learning models, encompassing 17 model families and more than 80 individual networks, including convolution-based architectures (`Starnet` [23], `FasterNet` [24], `MobileNetV4` [25], `EMO` [26], `ConvNeXtV2` [27]), transformer-based architectures (`Swin-Transformer` [28], `CSwin-Transformer` [29], `RepViT` [30], `EfficientViT` [31]), and YOLO variants (`YOLOv8` [32], `YOLOv9` [33], `YOLOv10` [34], `YOLOv11` [35], `YOLOv12` [36], `YOLOv13` [37], `Mamba-YOLO` [38], `Hyper-YOLO` [39]). These SOTA approaches cover a wide spectrum of design paradigms and represent the leading techniques in modern object detection and instance segmentation. Leveraging such architectural diversity allows us to establish a comprehensive benchmark while simultaneously validating the robustness and applicability of the dataset across different defect inspection scenarios.

For evaluation metrics, we utilize mean Average Precision (mAP) by following the widely adopted MS COCO [40] instance segmentation task, reporting bounding GT-Box mAP$_{0.5}$ and mAP$_{0.5:0.95}$ for objects localization, as well as mask mAP$_{0.5}$ and mAP$_{0.5:0.95}$ for objects segmentation (detailed in Section 3.2). Adopting these widely accepted metrics also aligns our dataset with common object-centric benchmarks, thereby enhancing its comparability and general applicability within the broader community.

3.1. Experimental Setup

All experiments—including both the benchmark evaluation on the CUBIT-InSeg dataset and cross-domain validation on external datasets—are conducted on an Ubuntu

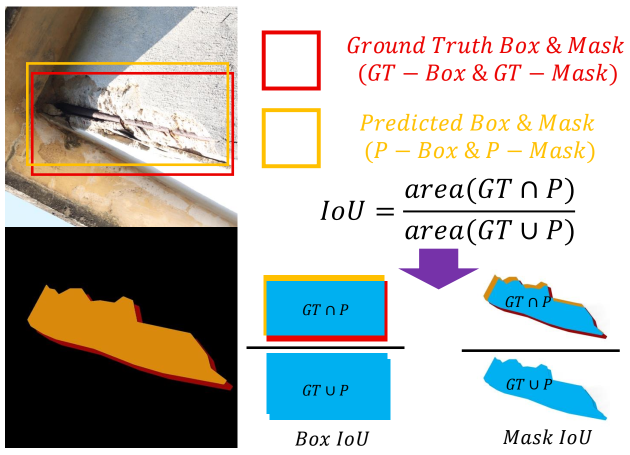

Ground Truth Box & Mask (GT − Box & GT − Mask)
Predicted Box & Mask (P − Box & P − Mask)
$IoU = \frac{area(GT \cap P)}{area(GT \cup P)}$
$GT \cap P$
$GT \cup P$
$GT \cap P$
$GT \cup P$
Box IoU
Mask IoU

Figure 6: Visualization of Intersection-over-Union (IoU). Red and orange represent the ground-truth bounding box / mask (GT-Box / GT-Mask) and predicted box / mask (P-Box / P-Mask) of this spalling sample, respectively. In IoU equation, the denominator symbolizes the union of the GT and the P, which is represented by a blue area. The overlapping area of the GT and the P, which denoted their intersection, is also indicated by the blue part.

22.04 workstation equipped with an Intel i9-13900K CPU and dual NVIDIA RTX 4090 GPUs. All models are trained for 300 epochs with a batch size of 16, and no pre-trained weights from any common object or defect-related datasets (e.g. [40, 41]) are used. Stochastic Gradient Descent (SGD) is adopted as the optimizer, and the detailed configurations of optimization parameters and hyperparameters are summarized in Table 3.

During inference, Non-Maximum Suppression (NMS) [42] is applied to remove redundant detections. For instance segmentation, although the segmentation mask is ultimately evaluated using mask-level IoU, the suppression process still relies on the GT-Box IoU, consistent with the procedure defined in Algorithm 1. Given an initial list of detected bounding boxes B and their confidence scores S, NMS iteratively selects the box with the highest confidence score, denoted as M, and adds it to the final detection set D. All remaining boxes $b_i \in B$ whose box IoU with M exceeds the suppression threshold N_t are removed along with their corresponding scores. Although mask-level IoU is later used to compute mask-based evaluation metrics such as Mask_AP$_{0.5}$ and Mask_AP$_{0.5:0.95}$, the suppression step remains box-based to ensure consistency and computational efficiency. These mask-based metrics evaluate the quality of the predicted segmentation masks by measuring their pixel-level agreement with the ground-truth shapes, thus reflecting the accuracy of instance-level defect delineation.

The IoU criterion evaluates the overlap ratio between a predicted box / mask (P-Box / P-Mask) and the ground-truth box / mask (GT-Box / GT-Mask). We adopt a confidence threshold of 0.25 and an IoU threshold of 0.7, which follows the setting of Ultratlytics framework. A visual illustration of the IoU of computation is provided in Figure 6.

Table 3
Optimizer and Hyperparameters of Experiments

Optimizer Type	SGD
Learning Rate Schedule	Linear
Initial Learning Rate α	1e-2
Final Learning Rate α	1e-4
Momentum β	0.937
Weight Decay ϕ	5e-4
Loss Coefficients λ_{cls}, λ_{box}, λ_{dfl}	0.5, 7.5 1.5

For benchmark training and testing our CUBIT-InSeg dataset, it is split into training (**5,596** images, **80**%), validation (**700** images, **10**%), and test (**700** images, **10**%) sets with resolution 640×640. Among the aforementioned SOTA models, both convolution-based and transformer-based architectures serve as powerful feature extraction backbones. To ensure a fair comparison, we integrated all these backbones into the Ultralytics[3], which is the framework used for YOLO series and its variants, and adopted the same Ultralytics neck and segmentation head, scaled using consistent multipliers (`n`, `s`, `m`, `l`, `x`). This unified implementation eliminates architectural discrepancies and allows performance differences to be attributed solely to the backbone design.

Algorithm 1: Non-maximum suppression (NMS) procedure used in instance segmentation pipeline

```
Input: The input initial detection boxes $B$, the corresponding
       confidence scores $S$, and the IoU suppression threshold
       $N_t$.
Output: The output final selected boxes $D$ and their
        corresponding scores $S$.
1  $D \leftarrow \varnothing$
2  while $B \neq empty$ do
3      Select the maximum value in the set of $S$, and give this
       value to $m$. $m \leftarrow argmax(S)$
4      $M \leftarrow b_m$
5      $D \leftarrow D \cup M$
6      $B \leftarrow B - M$
7      for $b_i$ in $B$ do
8          if $iou(M, b_i) > N_t$ then
9              $B \leftarrow B - b_i; S \leftarrow S - s_i$;
10 return $D$, $S$
```

3.2. Evaluation Metrics

We evaluate all models using the standard COCO evaluation protocol, which is applicable to both object detection and segmentation. For each predicted instance, the IoU between the prediction and its corresponding ground truth is computed—using bounding boxes for detection and pixel-level masks for segmentation. Based on IoU, a prediction is classified as a true positive (TP), false positive (FP), or false negative (FN).

The fundamental metrics *Precision* (P) and *Recall* (R) are defined as:

$$\text{Precision} = \frac{TP}{TP + FP}, \qquad \text{Recall} = \frac{TP}{TP + FN}, \tag{11}$$

[3]https://github.com/ultralytics/ultralytics

Table 4
Benchmark Results of Selected End-to-End Models on the Test Set of ***CUBIT-InSeg***

Models	#Params(M)↓	FLOPs(G)↓	FPS↑	Box_mAP$^{test}_{0.5}$↑	Box_mAP$^{test}_{0.5:0.95}$↑	Crack (Box)		Spalling (Box)		Mask_mAP$^{test}_{0.5}$↑	Mask_mAP$^{test}_{0.5:0.95}$↑	Crack (Mask)		Spalling (Mask)	
						AP$^{test}_{0.5}$↑	AP$^{test}_{0.5:0.95}$↑	AP$^{test}_{0.5}$↑	AP$^{test}_{0.5:0.95}$↑			AP$^{test}_{0.5}$↑	AP$^{test}_{0.5:0.95}$↑	AP$^{test}_{0.5}$↑	AP$^{test}_{0.5:0.95}$↑
Conv-based Models															
EMO-1M (n) [26]	3.27	13.1	498.35	84.5%	74.4%	71.1%	54.5%	97.9%	94.4%	76.1%	52.7%	54.7%	19.0%	97.4%	86.3%
EMO-2M (s) [26]	9.06	37.1	356.96	88.0%	78.7%	77.6%	61.9%	98.4%	95.5%	78.6%	55.3%	59.6%	22.4%	97.7%	88.2%
EMO-5M (m) [26]	20.46	86.1	242.75	$90.4\%_{2}$	82.2%	$82.0\%_{2}$	68.0%	98.8%	96.4%	$81.6\%_{1}$	57.5%	$64.9\%_{2}$	$25.2\%_{1}$	$98.3\%_{1}$	89.8%
EMO-6M (l) [26]	32.12	149.5	203.45	$90.5\%_{1}$	$82.4\%_{2}$	$82.1\%_{1}$	$68.2\%_{2}$	98.8%	$96.7\%_{1}$	81.2%	$57.9\%_{1}$	64.0%	25.0%	$98.3\%_{1}$	$90.7\%_{1}$
MobileNetV4-S (s) [25]	10.19	41.8	592.27	78.3%	66.4%	58.9%	40.8%	97.7%	92.0%	71.2%	48.5%	45.5%	13.3%	96.9%	83.7%
MobileNetV4-M (m) [25]	23.27	140.0	340.33	79.1%	67.6%	60.4%	42.3%	97.8%	93.0%	71.3%	48.6%	45.7%	13.7%	97.0%	83.4%
MobileNetV4-L (l) [25]	34.49	154.7	285.14	80.9%	70.0%	64.1%	46.1%	97.6%	93.8%	72.9%	50.5%	48.6%	14.8%	97.1%	86.1%
FasterNet-t0 (n) [24]	4.15	13.1	729.19_{2}	79.7%	68.2%	61.7%	43.2%	97.8%	93.1%	72.2%	49.0%	47.1%	14.1%	97.4%	84.0%
FasterNet-t1 (n) [24]	7.79	21.7	640.52	82.1%	70.6%	66.0%	47.4%	98.2%	93.9%	74.4%	50.3%	51.0%	16.0%	97.8%	84.7%
FasterNet-t2 (n) [24]	15.17	39.4	508.73	82.7%	71.3%	67.4%	48.9%	98.1%	93.7%	74.2%	50.6%	50.7%	15.5%	97.7%	85.8%
FasterNet-s (s) [24]	35.85	100.2	302.47	85.8%	78.0%	72.8%	54.2%	98.8%	95.3%	76.0%	52.3%	53.8%	16.9%	98.1%	87.7%
FasterNet-m (m) [24]	65.57	228.4	160.52	87.6%	77.3%	76.5%	58.8%	98.8%	95.8%	77.9%	53.9%	57.5%	19.2%	$98.2\%_{2}$	88.5%
FasterNet-l (l) [24]	109.29	349.8	124.97	88.4%	78.2%	78.0%	60.2%	$98.9\%_{2}$	96.2%	78.5%	54.5%	58.8%	19.9%	$98.3\%_{1}$	88.9%
Starnet-s50 (n) [23]	2.19_{1}	8.9_{1}	828.82_{1}	72.9%	61.0%	48.8%	31.9%	97.1%	90.0%	67.6%	46.0%	39.1%	11.2%	96.1%	80.8%
Starnet-s100 (n) [23]	2.69_{2}	10.4_{2}	689.58	76.8%	65.0%	56.0%	38.2%	97.5%	91.8%	69.9%	47.5%	43.1%	12.8%	96.7%	82.2%
Starnet-s150 (n) [23]	3.19	11.2	665.89	77.1%	65.1%	56.7%	38.5%	97.6%	91.7%	70.8%	47.8%	44.7%	13.2%	96.8%	82.4%
Starnet-s1 (s) [23]	8.45	30.8	480.88	82.3%	71.3%	66.5%	48.3%	98.1%	94.2%	74.8%	51.5%	52.1%	17.2%	97.5%	85.9%
Starnet-s2 (m) [23]	16.4	91.6	341.02	85.6%	75.4%	72.5%	55.1%	98.7%	95.6%	77.0%	53.4%	56.3%	18.4%	97.8%	88.3%
Starnet-s3 (l) [23]	21.7	104.3	287.39	86.0%	75.4%	73.3%	55.5%	98.8%	95.3%	77.4%	53.4%	56.8%	18.7%	98.0%	88.1%
Starnet-s4 (x) [23]	43.3	223.0	173.83	86.6%	76.1%	74.4%	56.7%	98.8%	95.5%	78.3%	54.1%	58.6%	19.8%	98.0%	88.4%
ConvNeXtV2-nano (n) [27]	17.91	47.8	210.86	87.6%	79.6%	77.4%	63.8%	97.8%	95.4%	79.1%	55.8%	60.7%	23.0%	97.4%	88.6%
ConvNeXtV2-tiny (s) [27]	35.36	97.7	136.16	87.8%	79.7%	77.5%	63.2%	98.2%	96.2%	79.3%	56.5%	60.7%	22.7%	97.8%	$90.4\%_{2}$
ConvNeXtV2-base (m) [27]	103.21	335.6	60.87	89.0%	81.0%	79.7%	65.9%	98.3%	96.2%	80.6%	57.0%	63.2%	24.3%	97.9%	89.7%
ConvNeXtV2-large (l) [27]	217.03	656.9	37.58	90.1%	$82.5\%_{1}$	81.8%	$68.4\%_{1}$	98.5%	$96.7\%_{1}$	$81.5\%_{2}$	$57.7\%_{2}$	$65.1\%_{1}$	$25.1\%_{2}$	98.0%	$90.4\%_{2}$
Transformer-based Models															
Swin-Transformer-Tiny (n) [28]	29.97	81.5	200.99	82.6%	72.3%	67.4%	50.4%	97.9%	94.2%	74.0%	51.9%	50.7%	17.2%	97.4%	86.5%
Swin-Transformer-Small (s) [28]	55.57	168.8	129.89	86.2%	76.2%	74.1%	57.0%	98.3%	95.3%	76.3%	53.7%	54.7%	18.9%	97.9%	88.5%
Swin-Transformer-Base (m) [28]	62.77	228.0	111.19	87.1%	77.6%	75.5%	59.0%	98.6%	96.1%	77.3%	54.6%	56.4%	19.8%	98.1%	89.5%
Swin-Transformer-Large (l) [28]	66.04	237.4	101.03	87.9%	78.3%	77.2%	60.9%	98.6%	95.8%	78.2%	55.0%	58.3%	20.4%	98.1%	89.6%
CSwin-Transformer-Tiny (n) [29]	23.94	74.4	210.68	81.8%	69.2%	66.3%	46.2%	97.2%	92.2%	72.9%	49.7%	49.1%	14.4%	96.8%	84.9%
CSwin-Transformer-Small (s) [29]	40.39	129.1	136.11	84.4%	72.8%	71.5%	52.3%	97.4%	93.3%	76.2%	52.1%	55.5%	17.6%	96.8%	86.7%
CSwin-Transformer-Base (m) [29]	90.53	318.7	72.24	86.1%	75.0%	74.7%	56.0%	97.5%	94.1%	77.3%	53.8%	57.9%	18.9%	96.8%	88.7%
CSwin-Transformer-Large (l) [29]	190.15	621.2	42.75	86.4%	75.3%	75.2%	56.7%	97.6%	93.9%	78.5%	54.5%	59.6%	20.1%	97.4%	88.9%
EfficientViT-M0 (n) [31]	3.99	11.8	641.55	79.1%	67.4%	60.3%	41.9%	97.9%	92.9%	71.6%	48.6%	45.7%	13.1%	97.5%	84.2%
EfficientViT-M1 (n) [31]	4.63	16.5	546.71	81.3%	69.3%	64.4%	45.4%	98.1%	93.3%	73.6%	50.1%	50.1%	15.1%	97.2%	85.1%
EfficientViT-M2 (s) [31]	9.81	35.2	474.59	84.2%	73.0%	70.0%	51.3%	98.3%	94.8%	76.0%	52.3%	54.2%	17.4%	97.8%	87.1%
EfficientViT-M3 (m) [31]	19.71	97.8	312.00	87.0%	76.7%	75.2%	57.6%	98.8%	95.9%	78.2%	54.4%	58.3%	19.8%	98.0%	89.0%
EfficientViT-M4 (l) [31]	24.88	109.2	264.32	87.3%	77.2%	75.7%	58.3%	98.8%	96.1%	78.7%	54.7%	59.2%	20.3%	$98.2\%_{2}$	89.0%
EfficientViT-M5 (x) [31]	48.60	236.1	172.96	89.0%	79.4%	79.1%	62.2%	$99.0\%_{1}$	$96.5\%_{2}$	80.0%	56.0%	61.8%	21.8%	$98.2\%_{2}$	90.2%
RepViT-m09 (n) [30]	6.68	20.9	504.95	82.5%	71.1%	66.9%	48.8%	98.0%	93.8%	75.2%	51.4%	53.0%	16.9%	97.4%	86.0%
RepViT-m10 (s) [30]	12.5	42.4	384.80	86.2%	75.9%	73.7%	56.3%	98.8%	95.4%	78.1%	54.3%	58.1%	20.4%	98.1%	88.3%
RepViT-m11 (m) [30]	21.1	105.2	294.08	87.0%	77.2%	75.2%	58.0%	$98.9\%_{2}$	96.4%	78.7%	54.8%	59.2%	20.6%	98.1%	89.0%
RepViT-m15 (l) [30]	30.2	129.7	239.38	87.3%	77.2%	75.9%	58.5%	98.6%	95.8%	78.8%	54.8%	59.5%	20.3%	98.1%	89.4%
RepViT-m23 (x) [30]	59.3	281.0	129.65	87.7%	77.5%	76.4%	59.1%	$98.9\%_{2}$	95.9%	79.2%	54.9%	60.2%	20.8%	98.1%	89.0%

(1) ↑ (↓) indicates that larger (smaller) values lead to better (worse) results. The best are in orange, the second best are in blue.

measuring the correctness of predictions and the ability to recover ground-truth defects, respectively.

To summarize performance across different recall levels, we adopt the COCO-style *Average Precision* (AP), computed by integrating the Precision–Recall curve. Following the COCO protocol, AP is evaluated at IoU threshold 0.5 ($AP_{0.5}$) and averaged across ten thresholds from 0.50 to 0.95 ($AP_{0.5:0.95}$).

$$AP = \int_0^1 p(r)\, dr, \tag{12}$$

and for multi-class tasks, the mAP is obtained by averaging AP over all defect categories:

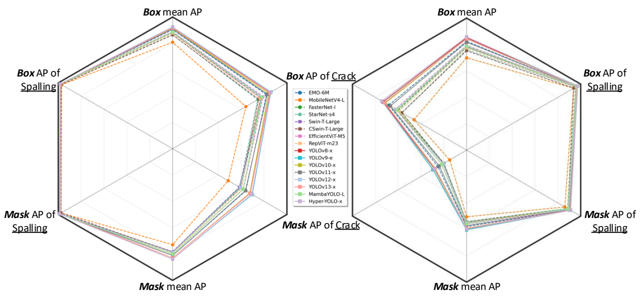
$$mAP = \frac{1}{n}\sum_{k=1}^{n} AP_k. \tag{13}$$

We report both Box_mAP (based on box IoU) and Mask_mAP (based on mask IoU). The box-based metrics evaluate localization accuracy, while the mask-based metrics assess pixel-level agreement and boundary fidelity—crucial for infrastructure defect segmentation.

3.3. Benchmarking Experiment and Analysis

3.3.1. Overall and Category-wise Performance Analysis of Instance Segmentation

Table 4 and 5 not only show the overall experimental results but also report of the per-defect-type evaluation results tested on our proposal CUBIT-InSeg dataset.



(a) ***IoU=0.5***, Average Precision Comparison of the Selected Models on our *CUBIT-InSeg* dataset

(b) ***IoU=0.95***, Average Precision Comparison of the Selected Models on our *CUBIT-InSeg* dataset

Figure 7: Comparison of six evaluation metrics at IoU thresholds of 0.5 (a) and 0.5:0.95 (b) using radar plots. Dashed curves correspond to End-to-End models, whereas solid curves correspond to Single-stage Real-time models.

As the sample data showed in Fig. 2, the detection and segmentation of the *cracks* are substantially more challenging than *spallings*: cracks typically appear thin, elongated, and fragmented, with ambiguous boundaries, whereas spalling regions are larger, more coherent, and visually well-defined. And the significant disparity of the number of targets is another factor (showed in Fig. 3). As illustrated in Fig. 7, the Box and Mask AP of Spalling lie much closer to the outer boundary of the radar plots compared with those for Crack, which means that the performance gap among models on spallings is relatively small, while cracks

Table 5
Benchmark Results of Selected Single-stage Real-time Models on the Test Set of ***CUBIT-InSeg***

Models	#Params(M)↓	FLOPs(G)↓	FPS↑	Box_mAP$^{test}_{0.5}$↑	Box_mAP$^{test}_{0.5:0.95}$↑	Crack (Box)		Spalling (Box)		Mask_mAP$^{test}_{0.5}$↑	Mask_mAP$^{test}_{0.5:0.95}$↑	Crack (Mask)		Spalling (Mask)	
						AP$^{test}_{0.5}$↑	AP$^{test}_{0.5:0.95}$↑	AP$^{test}_{0.5}$↑	AP$^{test}_{0.5:0.95}$↑			AP$^{test}_{0.5}$↑	AP$^{test}_{0.5:0.95}$↑	AP$^{test}_{0.5}$↑	AP$^{test}_{0.5:0.95}$↑
YOLO Series Models															
YOLOv8(8.1)-n [32]	3.25	12.0	835.53_2	82.9%	73.0%	68.3%	52.2%	97.5%	93.7%	75.3%	52.0%	53.7%	19.2%	96.9%	84.9%
YOLOv8(8.1)-s [32]	11.78	42.4	545.83	88.4%	79.2%	78.2%	62.6%	98.6%	95.8%	79.6%	55.7%	60.9%	22.7%	98.2%	88.8%
YOLOv8(8.1)-m [32]	27.22	111.0	311.14	90.5%	82.4%	82.3%	68.2%	98.6%	96.5%	81.6%	57.8%	65.2%	25.5%	98.0%	90.0%
YOLOv8(8.1)-l [32]	45.91	220.1	220.34	91.5%	83.8%	84.0%	70.7%	$99.0\%_2$	96.9%	82.6%	58.7%	66.7%	26.6%	$98.5\%_1$	90.7%
YOLOv8(8.1)-x [32]	71.72	343.7	145.91	92.0%	85.1%	85.1%	73.1%	98.8%	97.1%	83.0%	59.4%	67.9%	28.1%	98.2%	90.7%
YOLOv9-n [33]	3.09	54.1	536.10	80.7%	70.6%	63.7%	47.6%	97.8%	93.6%	74.0%	52.0%	50.8%	17.5%	97.3%	86.4%
YOLOv9-s [33]	8.52	75.4	408.28	87.6%	78.5%	76.9%	61.5%	98.3%	95.5%	79.6%	55.9%	61.3%	23.0%	97.9%	88.7%
YOLOv9-m [33]	22.25	131.2	275.07	91.0%	82.9%	83.2%	69.3%	98.8%	96.5%	81.4%	57.8%	64.9%	25.4%	98.0%	90.2%
YOLOv9-c [33]	27.62	157.6	235.83	91.9%	84.3%	85.0%	71.7%	98.8%	96.8%	82.5%	58.7%	66.8%	26.6%	98.2%	90.9%
YOLOv9-e [33]	59.68	244.4	128.57	$92.5\%_1$	$85.7\%_1$	$86.0\%_1$	$74.2\%_2$	98.9%	$97.3\%_1$	$83.9\%_1$	$60.4\%_1$	$69.6\%_1$	$29.5\%_1$	98.3%	$91.2\%_1$
YOLOv10-n [34]	2.84	11.7	836.54_1	84.4%	74.8%	71.2%	55.7%	97.7%	93.9%	77.5%	53.6%	57.5%	21.1%	94.4%	86.1%
YOLOv10-s [34]	9.17	40.5	530.40	89.3%	80.6%	80.1%	65.3%	98.5%	96.0%	80.7%	56.7%	63.3%	24.5%	98.0%	88.9%
YOLOv10-m [34]	19.33	101.4	305.73	90.7%	82.8%	82.8%	69.2%	98.7%	96.3%	82.6%	58.3%	67.0%	26.8%	98.2%	89.8%
YOLOv10-b [34]	25.48	166.6	254.87	90.9%	83.0%	83.2%	69.4%	98.6%	96.6%	82.2%	58.4%	66.4%	26.7%	98.1%	90.1%
YOLOv10-l [34]	30.79	194.9	224.79	90.8%	83.2%	82.7%	69.5%	98.9%	96.9%	82.3%	58.7%	66.2%	26.9%	$98.4\%_2$	90.5%
YOLOv10-x [34]	39.52	276.9	154.25	91.4%	84.2%	83.9%	71.3%	$99.0\%_2$	$97.2\%_2$	83.0%	59.3%	67.5%	27.9%	$98.4\%_2$	90.7%
YOLOv11(8.3)-n [35]	2.83	10.2_2	723.71	78.9%	66.8%	60.1%	41.2%	97.8%	92.4%	71.4%	48.3%	45.8%	13.5%	97.0%	83.1%
YOLOv11(8.3)-s [35]	10.07	35.3	466.25	85.8%	74.8%	73.2%	54.8%	98.3%	94.8%	77.8%	53.3%	57.7%	19.3%	98.0%	87.2%
YOLOv11(8.3)-m [35]	22.34	123.0	283.47	88.8%	78.5%	78.7%	61.3%	98.9%	95.7%	79.2%	55.0%	60.2%	21.2%	98.3%	88.8%
YOLOv11(8.3)-l [35]	27.59	141.9	240.49	89.1%	79.7%	79.3%	62.9%	98.9%	96.5%	80.0%	56.1%	61.7%	22.4%	98.3%	89.8%
YOLOv11(8.3)-x [35]	62.01	318.5	131.13	90.9%	81.8%	82.8%	66.8%	$99.1\%_1$	96.7%	81.1%	57.2%	63.7%	23.9%	$98.4\%_2$	90.4%
YOLOv12-n [36]	2.81_2	10.2_2	727.60	79.9%	68.2%	62.6%	44.0%	97.3%	92.5%	72.4%	49.2%	47.9%	14.6%	96.9%	83.9%
YOLOv12-s [36]	9.89	35.2	450.47	85.1%	74.1%	72.0%	53.6%	98.2%	94.7%	77.0%	52.5%	56.0%	18.4%	97.9%	86.6%
YOLOv12-m [36]	22.41	122.4	248.22	87.0%	76.2%	75.4%	57.0%	98.5%	95.3%	77.7%	53.7%	57.3%	19.1%	98.1%	88.4%
YOLOv12-l [36]	28.65	143.9	191.08	88.6%	78.5%	78.4%	60.8%	98.9%	96.3%	79.2%	55.0%	60.2%	20.8%	98.2%	89.2%
YOLOv12-x [36]	64.22	322.6	109.11	88.7%	78.9%	78.5%	61.5%	98.9%	96.4%	78.8%	55.2%	59.3%	20.6%	98.3%	89.7%
YOLOv13-n [37]	2.70_1	10.0_1	614.38	83.8%	75.4%	70.2%	56.7%	97.4%	94.1%	77.1%	53.9%	57.1%	21.2%	97.1%	86.5%
YOLOv13-s [37]	9.66	34.0	367.35	89.0%	81.6%	79.8%	66.9%	98.2%	96.3%	80.5%	57.0%	63.4%	25.3%	97.7%	88.8%
YOLOv13-l [37]	29.19	139.6	140.64	91.0%	84.0%	83.4%	71.2%	98.7%	96.8%	82.4%	58.9%	66.8%	27.2%	98.1%	90.5%
YOLOv13-x [37]	67.64	311.5	84.86	91.8%	84.3%	84.9%	71.9%	98.8%	96.7%	83.0%	59.1%	67.8%	27.5%	98.2%	$90.8\%_2$
Mamba-YOLO-T [38]	5.92	16.2	243.10	83.3%	72.3%	68.4%	50.2%	98.2%	94.5%	75.6%	52.2%	53.2%	17.5%	98.0%	87.0%
Mamba-YOLO-B [38]	21.15	58.5	121.19	87.5%	76.7%	76.0%	57.5%	98.9%	95.9%	77.2%	54.0%	55.8%	18.3%	$98.5\%_1$	89.8%
Mamba-YOLO-L [38]	56.33	175.9	45.08	88.5%	78.6%	78.0%	60.9%	98.9%	96.2%	80.0%	55.8%	61.6%	21.9%	$98.4\%_2$	88.8%
Hyper-YOLO-n [39]	3.87	13.4	742.44	85.4%	76.5%	72.8%	58.1%	98.1%	94.8%	77.6%	54.0%	57.4%	21.2%	97.7%	86.8%
Hyper-YOLO-s [39]	14.17	47.8	442.49	89.7%	81.4%	80.7%	66.4%	98.8%	96.5%	80.9%	56.7%	63.7%	24.2%	98.1%	89.2%
Hyper-YOLO-m [39]	32.08	123.1	249.75	91.5%	84.3%	84.3%	71.7%	98.8%	96.9%	82.9%	58.7%	67.5%	27.0%	98.2%	90.3%
Hyper-YOLO-l [39]	54.40	246.3	170.52	92.0%	$85.4\%_2$	85.3%	73.6%	98.8%	97.1%	$83.6\%_2$	59.7%	69.0%	28.2%	98.3%	$91.2\%_1$
Hyper-YOLO-x [39]	94.97	384.4	114.87	$92.4\%_2$	$85.7\%_1$	$85.8\%_2$	$74.3\%_1$	$99.0\%_2$	97.1%	$83.9\%_1$	$59.9\%_2$	$69.3\%_2$	$28.6\%_2$	$98.4\%_2$	$91.2\%_1$

(1) ↑ (↓) indicates that larger (smaller) values lead to better (worse) results. The best in orange, the second best are in blue.

exhibit noticeably lower AP values and contributes more significantly to the mAP degradation.

End-to-End Models As shown in Table 4, the conv-based architectures consistently outperform the transformer-based ones while also maintaining noticeably smaller model sizes. This performance advantage is largely driven by their superior ability to handle the more challenging *crack* category across all four mAP metrics. Benefiting from its conv-based *Expanded Window Multi-Head Self-Attention* (EW-MHSA) module, **`EMO-6M`** [26] ranks first on Box_mAP$_{0.5}$ and Mask_mAP$_{0.5:0.95}$, while **`EMO-5M`** achieves the top score on Mask_mAP$_{0.5}$. In addition, **`ConvNeXtV2-Large`** [27], equipped with the *Global Response Normalization* (GRN) layer, attains the best performance on Box_mAP$_{0.5:0.95}$.

For the *spalling* category, the performance differences among large models are marginal. Notably, the transformer-based **`EfficientViT-M5`** [31] performs competitively with conv-based models such as **`EMO-6M`** and **`FasterNet-l`** [24] on both Box and Mask_AP$_{0.5}$.

Single-stage Real-time Models As presented in Table 5, the YOLO family and its representative variants deliver strong performance despite their relatively compact model sizes. **`YOLOv9-e`** [33], supported by a *plug-and-play auxiliary training branch*, ranks first in three out of four *crack*-related AP metrics and ultimately achieves the best overall results across four box and mask mAP scores. The second-best performer is **`Hyper-YOLO-x`** [39], which benefits from the *Hypergraph Computation Empowered Semantic Collecting and Scattering* (HGCSCS) framework. **`Hyper-YOLO-x`** achieves the highest Box_mAP$_{0.5:0.95}$ on crack, but its Mask_mAP$_{0.5:0.95}$ is 0.9% lower than that of **`YOLOv9-e`**, which is a notable margin at such a stringent IoU threshold.

For the *spalling* category, performance differences are again small. At IoU=0.5, **`YOLOv11-x`** [35] yields the highest Box_AP, while **`Mamba-YOLO-M`** [38] and **`YOLOv8-l`** [32] share the best Mask_AP. At the more challenging IoU=0.5:0.95 setting, **`YOLOv9-e`** delivers the top Box_AP and jointly shares the best Mask_AP with **`Hyper-YOLO-l`** and **`Hyper-YOLO-x`**.

Comparison between End-to-End Models and Single-stage Real-time Models Comparing Table 4 with Table 5, the advantages of single-stage real-time models become substantially more evident. Under comparable model scales, single-stage architectures require fewer parameters and FLOPs, particularly when contrasted with complicated end-to-end architectures such as **`ConvNeXtV2`** [27], **`Swin-Transformer`** [28], and **`CSwin-Transformer`** [29]. In terms of instance segmentation accuracy, their superiority is even more pronounced. Across all four mAP metrics, the second-best single-stage model already surpasses the top-performing end-to-end model. The gap becomes striking under stricter IoU conditions: on Box_AP$_{0.5:0.95}$ and Mask_AP$_{0.5:0.95}$, the leading single-stage model (**`YOLOv9-e`**) exceeds the best end-to-end models (**`ConvNeXtV2-Large`** and **`EMO-6M`**) by 3.2% and 2.5%, respectively. This performance margin is primarily attributed to the markedly better handling of the more challenging *crack* category in both localization (Box-related metrics) and segmentation (Mask-related metrics).

To further illustrate these trends, Fig. 7 visualizes the strongest model from each series using a six-metric radar chart. The solid hexagons (single-stage models) consistently

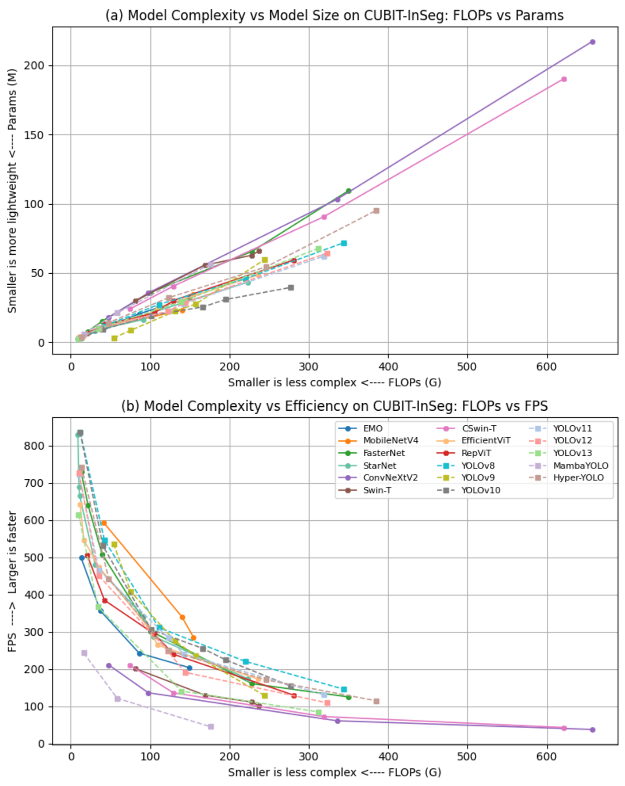


Figure 8: (a) Model Complexity (FLOPs) vs Model Size (Params), (b) Model Complexity (FLOPs) vs Inference Efficiency (FPS). Solid line correspond to End-to-End models, while dashed line represent Single-stage models.

envelop the dashed hexagons (end-to-end models), clearly indicating the uniformly stronger localization and segmentation performance achieved by single-stage models.

3.3.2. Network Attributes Affecting Instance Segmentation

The computational efficiency of an algorithm is another key factor for practical deployment. Table 4 and Table 5 summarize the model size (Params), computational complexity (FLOPs), and inference speed (FPS) of all evaluated architectures, as well. Params denotes the number of trainable parameters in a neural network, which also reflects the storage footprint of the trained model. FLOPs measure the computational cost required for a single forward pass, indicating how many floating-point operations the model must perform. FPS (frames per second) provides the most direct assessment of runtime inference efficiency.

In general, models with fewer parameters tend to require fewer FLOPs and are therefore expected to achieve higher FPS, as showed in Fig. 8. However, real-world inference performance is jointly influenced by several additional factors: **(1)** the extent to which the network architecture can fully utilize GPU Tensor Cores and CUDA kernels; **(2)** the level of hardware-specific optimization achieved by the deployment backend; **(3)** the computational overhead of post-processing steps such as NMS in Algorithm 1, which may increase when smaller inputs produce more small-object predictions. As a result, Params, FLOPs, and FPS should be interpreted together to provide a more reliable and comprehensive assessment of the practical efficiency of each model family.

End-to-End Models `StarNet` [23] series benefits from its lightweight design philosophy that relies on fewer layers and highly efficient element-wise multiplication. Consequently, `StarNet-s50` and `StarNet-s100` stand out as the two most compact models, each containing fewer than 3M parameters with only 8.9G and 10.4G FLOPs, respectively. As expected, `StarNet-s50` also delivers the highest inference speed, reaching 828.82 FPS on our Ubuntu 22.04 workstation. It is also noteworthy that `FasterNet-t0` [24], despite not being the smallest model, attains the second-highest throughput (729.19 FPS), owing to its partial convolution mechanism that significantly improves operator efficiency.

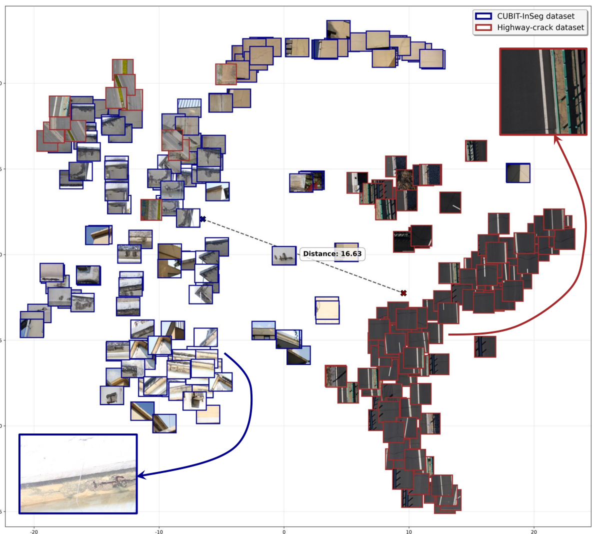
Single-stage Real-time Models Among all YOLO-family variants, the smallest architectures are `YOLOv13-n` [37] and `YOLOv12-n` [36], with Params of 2.70M and 2.81M, and FLOPs of 10.0G and 10.2G, respectively. Interestingly, the fastest models are not these smallest variants but rather `YOLOv10-n` [34] and `YOLOv8-n` [32], achieving 836.54 FPS and 835.53 FPS. In contrast, although `Mamba-YOLO-T` [38] has relatively small Params and FLOPs, its inference speed is considerably lower due to the *2D-Selective-Scan* [43], which introduces sequential dependencies.

Comparison between End-to-End and Single-stage Real-time Models Under the same model scaling factors, single-stage real-time architectures consistently exhibit lower Params and FLOPs while achieving substantially higher FPS, which can be more intuitively seen in Fig. 8. This efficiency gap is particularly evident when comparing with complex end-to-end models such as `ConvNeXtV2` [27], `Swin-Transformer` [28], `CSwin-Transformer` [29], highlighting the strong suitability of single-stage models for real-time defect inspection applications.

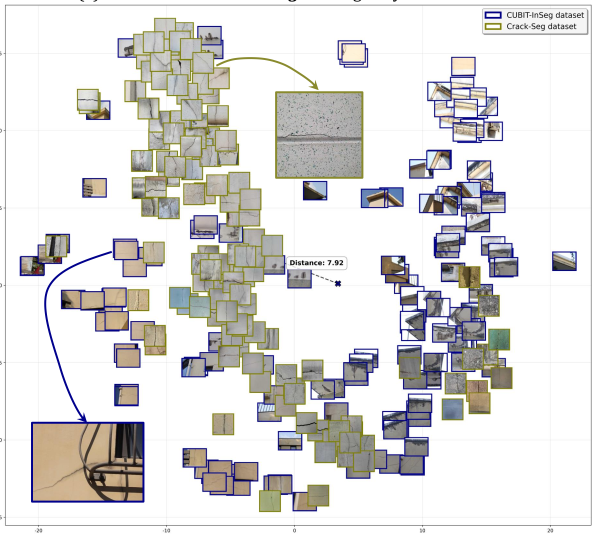
3.3.3. Zero-shot Validation on Cross-domain Datasets

After completing the benchmark evaluation of over 80 individual networks on the proposed CUBIT-InSeg dataset, we further assess the cross-domain generalization ability of our models under a zero-shot setting. Specifically, we directly apply the trained models to the test set parts of two publicly available unmanned systems collected datasets, Highway-Crack [15] for pure highway data and Crack-Seg [16] for building and pavement mixed data, without any training and fine-tuning.

To quantitatively and visually analyze the domain shift between datasets, Fig. 9 presents a t-SNE visualization computed from color-histogram descriptors. For each image, 32-bin RGB histograms are extracted, concatenated, and normalized to form a compact appearance descriptor. These histogram features are standardized and optionally compressed using Principal component analysis (PCA) for improved stability, and then projected into a two-dimensional embedding space via t-SNE. The resulting distributions reveal clear structural differences between CUBIT-InSeg and the external datasets. To further quantify the discrepancy, we



(a) t-SNE of our ***CUBIT-InSeg*** and Highway-Crack dataset



(b) t-SNE of our ***CUBIT-InSeg*** and Crack-Seg dataset

Figure 9: t-SNE visualization of our CUBIT-InSeg and (a) Highway-Crack [15] and (b) Crack-Seg [16] datasets.

compute the Euclidean distance between the cluster centers of the two t-SNE embeddings as a simple domain-gap metric. The distances are 16.63 for Highway-Crack and 7.92 for Crack-Seg. These discrepancies in feature-space geometry highlight the inherent challenges of achieving robust defect instance segmentation under diverse real-world conditions, while simultaneously demonstrating that models trained on our CUBIT-InSeg dataset retain strong adaptability across different infrastructure scenarios in the zero-shot setting.

Zero-shot on Highway-Crack Following the default evaluation protocol of Highway-Crack [15], we resize all input images to 512×512 during zero-shot testing. Table 6 reports the performance of the end-to-end models. Both **`EMO`** [26] and **`ConvNeXtV2`** [27] series continue to exhibit strong performance consistent with their results on our CUBIT-InSeg dataset. Notably, although **`CSwin-Transformer`** [29] performs moderately on our benchmark, it shows surprisingly strong generalization to the Highway-Crack dataset:

Table 6
Zero-shot Evaluation of Selected End-to-End Models on the Test Set of ***Highway-Crack*** [15]

Models	Input Size	Box_AP$^{test}_{0.5}$↑	Box_AP$^{test}_{0.5:0.95}$↑	Mask_AP$^{test}_{0.5}$↑	Mask_AP$^{test}_{0.5:0.95}$↑
Conv-based Models					
EMO-1M (n) [26]	512×512	80.9%	54.8%	64.8%	21.2%
EMO-2M (s) [26]	512×512	81.7%	56.8%	68.9%	23.5%
EMO-5M (m) [26]	512×512	83.1%$_2$	58.5%	72.5%	25.4%
EMO-6M (l) [26]	512×512	82.9%	59.7%	73.6%	26.6%
MobileNetV4Conv-S (s) [25]	512×512	78.0%	54.9%	63.8%	22.6%
MobileNetV4Conv-M (m) [25]	512×512	78.1%	55.5%	65.7%	22.8%
MobileNetV4Conv-L (l) [25]	512×512	79.3%	56.6%	68.2%	23.7%
FasterNet-t0 (n) [24]	512×512	77.4%	55.2%	64.3%	22.4%
FasterNet-t1 (n) [24]	512×512	78.9%	56.4%	68.5%	24.2%
FasterNet-t2 (n) [24]	512×512	79.8%	56.8%	69.8%	24.9%
FasterNet-s (s) [24]	512×512	81.4%	58.7%	72.2%	27.1%
FasterNet-m (m) [24]	512×512	82.6%	59.5%	74.0%$_2$	26.8%
FasterNet-l (l) [24]	512×512	82.7%	59.8%$_2$	72.5%	26.9%
Starnet-s50 (n) [23]	512×512	77.0%	53.6%	60.4%	19.7%
Starnet-s100 (n) [23]	512×512	78.1%	55.2%	64.9%	22.7%
Starnet-s150 (n) [23]	512×512	77.1%	54.9%	65.8%	22.3%
Starnet-s1 (s) [23]	512×512	79.2%	56.8%	69.4%	23.7%
Starnet-s2 (m) [23]	512×512	81.2%	58.3%	71.4%	26.5%
Starnet-s3 (l) [23]	512×512	80.7%	57.7%	71.7%	26.3%
Starnet-s4 (x) [23]	512×512	82.2%	59.2%	71.0%	26.1%
ConvNeXtV2-nano (n)	512×512	78.7%	56.5%	69.4%	26.3%
ConvNeXtV2-tiny (s)	512×512	78.8%	56.9%	69.3%	25.5%
ConvNeXtV2-base (m)	512×512	78.1%	57.1%	70.0%	27.1%
ConvNeXtV2-large (l)	512×512	79.7%	58.3%	73.5%	27.9%$_2$
Transformer-based Models					
Swin-Transformer-Tiny (n) [28]	512×512	74.8%	53.5%	62.8%	21.9%
Swin-Transformer-Small (s) [28]	512×512	76.9%	55.0%	66.0%	24.3%
Swin-Transformer-Base (m) [28]	512×512	78.5%	56.6%	69.9%	26.5%
Swin-Transformer-Large (l) [28]	512×512	79.1%	57.3%	69.3%	26.9%
CSwin-Transformer-Tiny (n) [29]	512×512	78.7%	56.5%	69.4%	24.7%
CSwin-Transformer-Small (s) [29]	512×512	81.3%	58.1%	71.4%	26.5%
CSwin-Transformer-Base (m) [29]	512×512	81.8%	59.1%	73.1%	27.7%
CSwin-Transformer-Large (l) [29]	512×512	82.2%	59.7%	73.7%	27.8%
EfficientViT-M0 (n) [31]	512×512	78.2%	55.5%	66.3%	23.1%
EfficientViT-M1 (n) [31]	512×512	79.1%	56.6%	68.2%	23.7%
EfficientViT-M2 (s) [31]	512×512	80.6%	58.1%	68.5%	24.4%
EfficientViT-M3 (m) [31]	512×512	81.8%	59.0%	72.4%	26.6%
EfficientViT-M4 (l) [31]	512×512	81.8%	59.3%	73.8%	27.8%
EfficientViT-M5 (x) [31]	512×512	83.3%$_1$	60.7%$_1$	74.7%$_1$	28.2%$_1$
RepViT-m09 (n) [30]	512×512	79.3%	56.6%	69.7%	23.9%
RepViT-m10 (s) [30]	512×512	80.8%	58.4%	71.1%	25.7%
RepViT-m11 (m) [30]	512×512	80.9%	58.5%	71.7%	25.9%
RepViT-m15 (l) [30]	512×512	80.5%	58.7%	71.2%	26.6%
RepViT-m23 (x) [30]	512×512	81.5%	59.0%	71.7%	27.2%

(1) ↑ (↓) indicates that larger (smaller) values lead to better (worse) results. The best in orange, the second best are in blue.

`CSwin-Transformer-Base` ranks second on three metrics, and **`CSwin-Transformer-Large`** achieves the highest Box_AP$_{0.5:0.95}$.

Table 7 presents the zero-shot performance of the single-stage real-time models. Besides the consistently strong **`YOLOv9`** [33] and **`Hyper-YOLO`** [39], both **`YOLOv8`** [32] and **`YOLOv10`** [34] also demonstrate competitive cross-domain robustness. **`YOLOv8-l`** and **`YOLOv8-x`** obtain the highest Mask_AP$_{0.5}$ and rank second in Box_AP$_{0.5}$, while **`YOLOv10-b`** and **`YOLOv10-l`** achieve the best Box_AP$_{0.5}$.

Overall, combining the results from Table 6 and Table 7, it is obvious that for UAV-perspective highway-crack imagery, characterized by cluttered backgrounds and small, distant defects, larger-capacity models (**`m`**, **`l`**, **`x`**) consistently demonstrate stronger zero-shot robustness across both end-to-end and single-stage architectures.

Zero-shot on Crack-Seg For the Crack-Seg dataset, we follow its default setting and resize all input images to 416×416 during zero-shot testing. Table 8 summarizes the results of the end-to-end models. **`CSwin-Transformer-Large`** [29] and **`EfficientViT-M5`** [31] achieve the highest Box_AP$_{0.5}$ and Box_AP$_{0.5:0.95}$, respectively. Interestingly, despite being the smallest variant, **`CSwin-Transformer-Tiny`** obtains the best performance on two mask-based metrics, demonstrating that compact transformer architectures can generalize effectively under certain cross-domain conditions.

Table 7
Zero-shot Evaluation of the Selected Single-stage Real-time Models on the Test Set of ***Highway-Crack*** [15]

Models	Input Size	Box_AP$^{test}_{0.5}$↑	Box_AP$^{test}_{0.5:0.95}$↑	Mask_AP$^{test}_{0.5}$↑	Mask_AP$^{test}_{0.5:0.95}$↑
YOLO Series Models					
YOLOv8(8.1)-n [32]	512×512	79.9%	55.0%	64.3%	21.8%
YOLOv8(8.1)-s [32]	512×512	81.3%	56.9%	67.4%	23.6%
YOLOv8(8.1)-m [32]	512×512	82.8%	59.0%	72.7%	26.6%
YOLOv8(8.1)-l [32]	512×512	84.7%$_2$	60.8%	76.4%	28.5%
YOLOv8(8.1)-x [32]	512×512	84.3%	61.3%$_2$	76.8%$_2$	29.5%
YOLOv9-t [33]	512×512	81.0%	56.4%	66.0%	22.8%
YOLOv9-s [33]	512×512	83.3%	58.8%	71.4%	24.7%
YOLOv9-m [33]	512×512	84.3%	60.3%	75.0%	27.6%
YOLOv9-c [33]	512×512	84.4%	60.4%	75.2%	28.7%
YOLOv9-e [33]	512×512	85.3%$_1$	62.0%$_1$	77.2%$_1$	30.4%$_1$
YOLOv10-n [34]	512×512	80.9%	56.7%	66.1%	22.4%
YOLOv10-s [34]	512×512	83.6%	58.7%	71.5%	24.8%
YOLOv10-m [34]	512×512	83.3%	60.5%	75.3%	27.5%
YOLOv10-b [34]	512×512	84.4%	60.1%	74.5%	27.6%
YOLOv10-l [34]	512×512	84.3%	60.5%	74.7%	28.0%
YOLOv10-x [34]	512×512	84.5%	61.2%	75.2%	28.8%
YOLOv11(8.3)-n [35]	512×512	77.9%	55.0%	65.0%	21.9%
YOLOv11(8.3)-s [35]	512×512	80.8%	57.9%	70.0%	24.6%
YOLOv11(8.3)-m [35]	512×512	81.4%	58.3%	72.2%	26.4%
YOLOv11(8.3)-l [35]	512×512	82.8%	59.9%	73.7%	26.8%
YOLOv11(8.3)-x [35]	512×512	83.5%	60.5%	75.5%	28.6%
YOLOv12-n [36]	512×512	78.7%	56.1%	65.3%	22.4%
YOLOv12-s [36]	512×512	81.1%	58.4%	70.3%	24.5%
YOLOv12-m [36]	512×512	81.6%	59.0%	73.7%	27.1%
YOLOv12-l [36]	512×512	82.4%	59.6%	74.8%	27.5%
YOLOv12-x [36]	512×512	83.0%	59.9%	74.7%	27.6%
YOLOv13-n [37]	512×512	76.8%	54.6%	66.2%	23.3%
YOLOv13-s [37]	512×512	79.2%	57.3%	71.7%	26.7%
YOLOv13-l [37]	512×512	82.3%	59.7%	73.9%	28.6%
YOLOv13-x [37]	512×512	82.2%	60.3%	75.1%	29.2%
MambaYOLO-T [38]	512×512	79.0%	57.7%	69.2%	25.4%
MambaYOLO-B [38]	512×512	81.7%	59.7%	75.0%	28.2%
MambaYOLO-L [38]	512×512	82.1%	60.3%	74.6%	29.8%$_2$
Hyper-YOLO-n [39]	512×512	78.3%	55.6%	65.1%	22.3%
Hyper-YOLO-s [39]	512×512	81.0%	58.6%	69.9%	25.0%
Hyper-YOLO-m [39]	512×512	82.8%	60.2%	73.7%	27.5%
Hyper-YOLO-l [39]	512×512	83.1%	61.0%	75.4%	29.3%
Hyper-YOLO-x [39]	512×512	83.6%	61.3%$_2$	75.5%	29.5%

(1) ↑ (↓) indicates that larger (smaller) values lead to better (worse) results. The best in orange, the second best are in blue.

Table 8
Zero-shot Evaluation of Selected End-to-End Models on the Test Set of ***Crack-Seg*** [32]

Models	Input Size	Box_AP$^{test}_{0.5}$↑	Box_AP$^{test}_{0.5:0.95}$↑	Mask_AP$^{test}_{0.5}$↑	Mask_AP$^{test}_{0.5:0.95}$↑
Conv-based Models					
EMO-1M (n) [26]	416×416	72.4%	60.1%	63.5%	28.2%
EMO-2M (s) [26]	416×416	76.5%	62.6%	61.9%	29.2%
EMO-5M (m) [26]	416×416	75.6%	64.3%	68.9%	29.8%
EMO-6M (l) [26]	416×416	74.9%	62.3%	66.7%	29.1%
MobileNetV4Conv-S (s) [25]	416×416	71.8%	61.6%	66.0%	28.3%
MobileNetV4Conv-M (m) [25]	416×416	72.6%	62.5%	63.9%	29.1%
MobileNetV4Conv-L (l) [25]	416×416	75.0%	62.7%	66.5%	30.5%
FasterNet-t0 (n) [24]	416×416	72.6%	63.3%	65.4%	27.8%
FasterNet-t1 (n) [24]	416×416	72.4%	62.2%	64.7%	27.4%
FasterNet-t2 (n) [24]	416×416	70.9%	60.8%	63.2%	27.9%
FasterNet-s (s) [24]	416×416	71.6%	62.5%	66.4%	30.2%
FasterNet-m (m) [24]	416×416	72.5%	62.1%	63.6%	30.1%
FasterNet-l (l) [24]	416×416	77.6%$_2$	65.4%	69.8%	31.5%
Starnet-s50 (n) [23]	416×416	65.8%	53.7%	54.2%	18.5%
Starnet-s100 (n) [23]	416×416	72.8%	63.0%	64.9%	28.6%
Starnet-s150 (n) [23]	416×416	70.2%	61.2%	63.5%	28.1%
Starnet-s1 (s) [23]	416×416	71.9%	62.7%	67.0%	31.5%
Starnet-s2 (m) [23]	416×416	73.5%	63.1%	63.8%	30.2%
Starnet-s3 (l) [23]	416×416	77.4%	64.0%	66.8%	31.2%
Starnet-s4 (x) [23]	416×416	76.5%	65.2%	67.1%	31.7%
ConvNeXt2-nano (n) [27]	416×416	73.9%	63.4%	68.2%	31.7%
ConvNeXt2-tiny (s) [27]	416×416	73.0%	62.2%	68.3%	31.2%
ConvNeXt2-base (m) [27]	416×416	75.6%	64.7%	66.7%	32.4%
ConvNeXt2-large (l) [27]	416×416	75.8%	65.6%$_2$	69.3%	32.8%$_2$
Transformer-based Models					
Swin-Transformer-Tiny (n)	512×512	75.0%	63.7%	70.9%$_1$	33.7%$_1$
Swin-Transformer-Small (s)	512×512	76.8%	63.9%	68.9%	31.9%
Swin-Transformer-Base (m)	512×512	75.8%	64.1%	69.2%	31.9%
Swin-Transformer-Large (l)	512×512	76.7%	64.9%	65.7%	31.0%
CSwin-Transformer-Tiny (n) [29]	416×416	72.7%	63.1%	69.3%	30.6%
CSwin-Transformer-Small (s) [29]	416×416	74.8%	63.9%	69.9%	31.7%
CSwin-Transformer-Base (m) [29]	416×416	75.1%	63.6%	69.3%	32.4%
CSwin-Transformer-Large (l) [29]	416×416	77.9%$_1$	65.6%$_2$	70.1%$_2$	32.2%
EfficientViT-M0 (n) [31]	416×416	72.3%	63.1%	64.7%	28.3%
EfficientViT-M1 (n) [31]	416×416	72.5%	63.2%	65.9%	29.2%
EfficientViT-M2 (s) [31]	416×416	71.3%	62.1%	66.1%	28.9%
EfficientViT-M3 (m) [31]	416×416	73.0%	63.0%	64.0%	31.0%
EfficientViT-M4 (l) [31]	416×416	74.1%	62.7%	66.6%	30.3%
EfficientViT-M5 (x) [31]	416×416	77.1%	66.0%$_1$	66.5%	31.6%
RepViT-m09 (n) [30]	416×416	72.7%	62.5%	64.1%	28.7%
RepViT-m10 (s) [30]	416×416	71.8%	62.9%	67.2%	30.9%
RepViT-m11 (m) [30]	416×416	73.5%	63.3%	64.0%	30.8%
RepViT-m11 (l) [30]	416×416	76.7%	64.3%	68.4%	30.7%
RepViT-m23 (x) [30]	416×416	74.2%	63.9%	69.8%	30.8%

(1) ↑ (↓) indicates that larger (smaller) values lead to better (worse) results. The best in orange, the second best are in blue.

Table 9 shows the results for single-stage real-time models. Under IoU=0.5, **`Mamba-YOLO-L`** [38] and **`YOLOv10-x`** [34] deliver the highest Box_AP and Mask_AP, respectively. Under the more demanding IoU (from 0.5 to 0.95), **`YOLOv13-x`** [37], despite its relatively modest in-domain performance, outperforms all competitors, achieving the best Box_AP and Mask_AP, with the latter being 0.22% higher than the second-best model **`YOLOv9-s`** [33].

Taken together, Tables 8 and 9 indicate that, unlike Highway-Crack, the Crack-Seg dataset does not strictly follow the trend where larger models generalize better. Several lightweight models (**`n`**, **`s`**), such as **`CSwin-Transformer-Tiny`**, **`YOLOv9-s`**, **`YOLOv10-s`**, and **`Hyper-YOLO-s`**, achieve top-two performance on multiple metrics, revealing a different behavior for this cross-domain ground vehicle-captured data.

Results Comparison between the Two Cross-domain Datasets Although Fig. 9 suggests that Crack-Seg is closer to our CUBIT-InSeg dataset in terms of t-SNE distribution similarity, the zero-shot instance segmentation performance on Crack-Seg is slightly inferior to that on Highway-Crack. At IoU=0.5, most models achieve over 80% Box_AP and 75% Mask_AP on Highway-Crack, whereas the corresponding metrics on Crack-Seg marginally reach these levels (see Fig. 10(a) and (c)). At a more stringent IoU=0.5:0.95, models exhibit a wide performance spread on Highway-Crack, where strong models achieve markedly higher AP scores while weaker ones drop substantially, indicating that this dataset is more sensitive to model capability. In contrast, the Box_AP and Mask_AP results on Crack-Seg are more concentrated with a narrower performance range, suggesting that its segmentation difficulty is more uniformly distributed across different architectures.

Several factors contribute to this disxicrepancy: **(1)** Crack-Seg uses a smaller default input resolution of 416 × 416, increasing the scale gap relative to the 640 × 640 training resolution used for CUBIT-InSeg than Highway-Crack's input resolution; **(2)** Crack-Seg's defect targets are tiny and lie near image borders, making segmentation particularly difficult under reduced input resolution; **(3)** Crack-Seg contains a mixture of viewpoints, including substantial non-UAV perspectives, close-range roadway and building imagery, introducing a domain shift that is larger in semantics than what the t-SNE histogram descriptors can fully capture. In summary, these observations collectively explain why Crack-Seg exhibits a smaller distribution distance in histogram-based t-SNE space, yet yields comparatively poorer zero-shot segmentation performance.

In addition, when comparing different model families, we observe that although transformer-based architectures may underperform conv-based models when trained from

Table 9
Zero-shot Evaluation of the Selected Single-stage Real-time Models on the Test Set of ***Crack-Seg*** [16]

Models	Input Size	Box_AP$^{test}_{0.5}$↑	Box_AP$^{test}_{0.5:0.95}$↑	Mask_AP$^{test}_{0.5}$↑	Mask_AP$^{test}_{0.5:0.95}$↑
YOLO Series Models					
YOLOv8(8.1)-n [32]	416×416	75.6%	64.2%	69.9%	31.7%
YOLOv8(8.1)-s [32]	416×416	74.4%	63.8%	69.2%	29.7%
YOLOv8(8.1)-m [32]	416×416	74.2%	64.3%	70.0%	32.0%
YOLOv8(8.1)-l [32]	416×416	74.3%	62.3%	68.0%	33.5%
YOLOv8(8.1)-x [32]	416×416	75.3%	62.6%	67.8%	31.2%
YOLOv9-t [33]	416×416	76.6%	64.1%	69.7%	31.5%
YOLOv9-s [33]	416×416	76.3%	65.2%	71.0%	33.9%$_2$
YOLOv9-m [33]	416×416	73.9%	62.4%	67.6%	30.5%
YOLOv9-c [33]	416×416	73.8%	60.8%	61.6%	30.3%
YOLOv9-e [33]	416×416	75.5%	62.1%	69.3%	31.0%
YOLOv10-n [34]	416×416	75.3%	62.9%	68.1%	30.4%
YOLOv10-s [34]	416×416	75.6%	66.5%$_2$	71.9%$_2$	33.2%
YOLOv10-m [34]	416×416	73.0%	61.4%	69.9%	32.3%
YOLOv10-b [34]	416×416	75.5%	64.4%	67.0%	31.5%
YOLOv10-l [34]	416×416	77.5%	64.4%	70.1%	32.1%
YOLOv10-x [34]	416×416	77.0%	64.7%	73.5%$_1$	33.0%
YOLOv11(8.3)-n [35]	416×416	73.0%	63.0%	63.8%	29.1%
YOLOv11(8.3)-s [35]	416×416	71.9%	63.1%	67.2%	32.1%
YOLOv11(8.3)-m [35]	416×416	73.6%	63.5%	64.2%	30.6%
YOLOv11(8.3)-l [36]	416×416	76.6%	65.0%	68.6%	32.1%
YOLOv11(8.3)-x [36]	416×416	75.6%	64.1%	68.8%	31.2%
YOLOv12-n [36]	416×416	72.5%	62.4%	65.4%	29.2%
YOLOv12-s [36]	416×416	72.1%	63.1%	67.8%	31.7%
YOLOv12-m [36]	416×416	73.9%	63.0%	64.2%	31.4%
YOLOv12-l [36]	416×416	76.3%	65.6%	67.6%	31.4%
YOLOv12-x [36]	416×416	76.0%	64.6%	67.8%	31.4%
YOLOv13-n [37]	416×416	75.2%	63.2%	64.2%	30.0%
YOLOv13-s [37]	416×416	75.6%	62.8%	67.1%	31.5%
YOLOv13-l [37]	416×416	76.1%	66.1%	70.6%	33.3%
YOLOv13-x [37]	416×416	78.7%$_2$	66.7%$_1$	70.7%	36.1%$_1$
MambaYOLO-T [38]	416×416	73.0%	63.3%	65.1%	29.2%
MambaYOLO-B [38]	416×416	73.0%	62.7%	63.4%	28.8%
MambaYOLO-L [38]	416×416	79.1%$_1$	65.7%	71.4%	33.8%
Hyper-YOLO-n [39]	416×416	75.6%	64.4%	68.0%	31.1%
Hyper-YOLO-s [39]	416×416	75.1%	65.7%	71.9%$_2$	32.6%
Hyper-YOLO-m [39]	416×416	72.1%	61.4%	63.1%	30.3%
Hyper-YOLO-l [39]	416×416	75.0%	65.2%	69.1%	31.9%
Hyper-YOLO-x [39]	416×416	72.5%	61.8%	66.5%	31.7%

(1) ↑ (↓) indicates that larger (smaller) values lead to better (worse) results. The best in orange, the second best are in blue.

scratch on our CUBIT-InSeg dataset, they demonstrate notably strong zero-shot generalization on both Highway-Crack and Crack-Seg datasets. This further highlights the robustness and cross-domain adaptability of transformer-based models, particularly when facing variations in image resolution, viewpoint, and underlying data distributions.

4. Conclusion

CRediT authorship contribution statement

Benyun Zhao: Conceptualization, Investigation, Formal analysis, Writing - Original Draft. **Jihan Zhang:** Conceptualization, Investigation, Formal analysis, Writing - Original Draft. **Guidong Yang:** Conceptualization, Investigation, Formal analysis, Writing - Original Draft. **Yijun Huang:** Method and Dataset Investigation, Formal analysis, Writing - Original Draft. **Lei Lei:** Hardware Platform, Real-world Experiment, Writing - Review & Editing. **Xi Chen:** Funding acquisition, Supervision, Writing - Review & Editing, Project administration. **Ben M. Chen:** Conceptualization, Funding acquisition, Resources, Supervision, Writing - Review & Editing, Project administration.

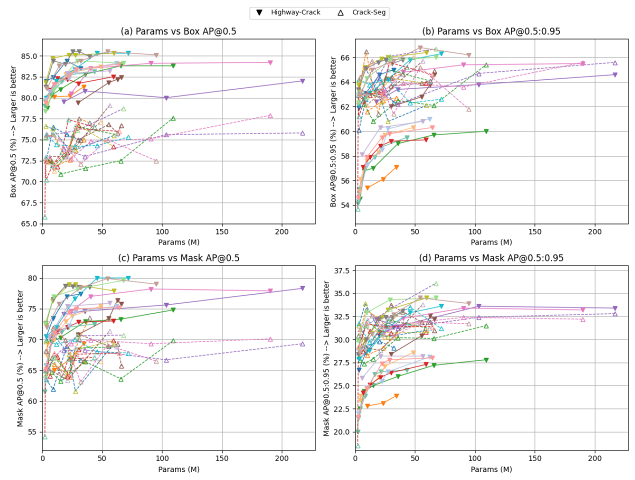


Figure 10: Zero-shot instance segmentation results based on models trained on the proposed CUBIT-InSeg dataset, evaluated on the Highway-Crack and Crack-Seg test sets. Solid lines with inverted triangles ▽ denote Highway-Crack, while dashed lines with upright triangles △ denote Crack-Seg. (a) Box_AP$_{0.5}$, (b) Box_AP$_{0.5:0.95}$, (c) Mask_AP$_{0.5}$, and (d) Mask_AP$_{0.5:0.95}$.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in the paper.

Acknowledgments

This work is supported by the InnoHK of the Government of the Hong Kong Special Administrative Region via the Hong Kong Centre for Logistics Robotics (HKCLR).

References

[1] Junjie Chen, Isabelle Chan, and Ioannis Brilakis. Shifting research from defect detection to defect modeling in computer vision-based structural health monitoring. *Automation in Construction*, 164:105481, 2024.

[2] Krisada Chaiyasarn, Apichat Buatik, Hisham Mohamad, Mingliang Zhou, Sirisilp Kongsilp, and Nakhorn Poovarodom. Integrated pixel-level cnn-fcn crack detection via photogrammetric 3d texture mapping of concrete structures. *Automation in Construction*, 140:104388, 2022.

[3] Gustavo H Beckman, Dimos Polyzois, and Young-Jin Cha. Deep learning-based automatic volumetric damage quantification using depth camera. *Automation in Construction*, 99:114–124, 2019.

[4] Fardin Bahreini and Amin Hammad. Dynamic graph cnn based semantic segmentation of concrete defects and as-inspected modeling. *Automation in Construction*, 159:105282, 2024.

[5] Zehao Ye, Lucy Lovell, Asaad Faramarzi, and Jelena Ninić. Sam-based instance segmentation models for the automation of structural damage detection. *Advanced Engineering Informatics*, 62:102826, 2024.

[6] Wenjun Wang and Chao Su. Optimizing concrete surface defect detection with adaptive supervision and scribble annotations. *Advanced Engineering Informatics*, 68:103669, 2025.

[7] Jiepeng Liu, Zhengtao Yang, Hongtuo Qi, Tong Jiao, Dongsheng Li, Zhou Wu, Nina Zheng, and Shaoqian Xu. Deep learning-assisted

automatic quality assessment of concrete surfaces with cracks and bugholes. *Advanced Engineering Informatics*, 62:102577, 2024.

[8] Seongdeok Bang, Somin Park, Hongjo Kim, and Hyoungkwan Kim. Encoder–decoder network for pixel-level road crack detection in black-box images. *Computer-Aided Civil and Infrastructure Engineering*, 34(8):713–727, 2019.

[9] Shang Jiang and Jian Zhang. Real-time crack assessment using deep neural networks with wall-climbing unmanned aerial system. *Computer-Aided Civil and Infrastructure Engineering*, 35(6):549–564, 2020.

[10] Junjie Chen, Weisheng Lu, and Jinfeng Lou. Automatic concrete defect detection and reconstruction by aligning aerial images onto semantic-rich building information model. *Computer-Aided Civil and Infrastructure Engineering*, 38(8):1079–1098, 2023.

[11] Jihan Zhang, Benyun Zhao, Guidong Yang, Xunkuai Zhou, Yijun Huang, Chuanxiang Gao, Xi Chen, and Ben M Chen. Ai-empowered digital twin modeling for high-precision building defect management integrating uav and geobim. In *Building Simulation*, pages 1–28. Springer, 2025.

[12] Fan Yang, Lei Zhang, Sijia Yu, Danil Prokhorov, Xue Mei, and Haibin Ling. Feature pyramid and hierarchical boosting network for pavement crack detection. *IEEE Transactions on Intelligent Transportation Systems*, 21(4):1525–1535, 2019.

[13] Qipei Mei, Mustafa Gül, and Md Riasat Azim. Densely connected deep neural network considering connectivity of pixels for automatic crack detection. *Automation in Construction*, 110:103018, 2020.

[14] Ronny Stricker, Dustin Aganian, Maximilian Sesselmann, Daniel Seichter, Marius Engelhardt, Roland Spielhofer, Matthias Hahn, Astrid Hautz, Klaus Debes, and Horst-Michael Gross. Road surface segmentation-pixel-perfect distress and object detection for road assessment. In *2021 IEEE 17th International Conference on Automation Science and Engineering (CASE)*, pages 1789–1796. IEEE, 2021.

[15] Zhonghua Hong, Fan Yang, Haiyan Pan, Ruyan Zhou, Yun Zhang, Yanling Han, Jing Wang, Shuhu Yang, Peng Chen, Xiaohua Tong, et al. Highway crack segmentation from unmanned aerial vehicle images using deep learning. *IEEE Geoscience and Remote Sensing Letters*, 19:1–5, 2021.

[16] University. crack dataset, dec 2022. visited on 2024-01-23.

[17] Christian Benz, Paul Debus, Huy Khanh Ha, and Volker Rodehorst. Crack segmentation on uas-based imagery using transfer learning. In *2019 International Conference on Image and Vision Computing New Zealand (IVCNZ)*, pages 1–6, 2019.

[18] Xinyi Wang, Chuanxiang Gao, Xi Chen, and Ben M. Chen. Fast and secure distributed multi-agent coverage control with an application to infrastructure inspection and reconstruction. In *Proceedings of the 42nd Chinese Control Conference*, pages 5998–6003. IEEE, 2023.

[19] Chuanxiang Gao, Xinyi Wang, Xi Chen, and Ben M. Chen. A hierarchical multi-UAV cooperative framework for infrastructure inspection and reconstruction. *Control Theory and Technology*, 22(3):394–405, 2024.

[20] Buildings and constructed assets – service life planning – part 7: Performance evaluation for feedback of service life data from practice, 2017. British Standards Institution adoption of ISO standard.

[21] Buildings Department. Guidelines on mandatory building inspection scheme, 2017. Volume 1: Pre-1980 Residential and Composite Buildings.

[22] Hong Kong Institute of Surveyors. Practice notes for building surveying and façade inspection, 2016.

[23] Xu Ma, Xiyang Dai, Yue Bai, Yizhou Wang, and Yun Fu. Rewrite the stars. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024.

[24] Jierun Chen, Shiu-hong Kao, Hao He, Weipeng Zhuo, Song Wen, Chul-Ho Lee, and S-H Gary Chan. Run, don't walk: Chasing higher flops for faster neural networks. *arXiv preprint arXiv:2303.03667*, 2023.

[25] Danfeng Qin, Chas Leichner, Manolis Delakis, Marco Fornoni, Shixin Luo, Fan Yang, Weijun Wang, Colby Banbury, Chengxi Ye, Berkin Akin, et al. Mobilenetv4-universal models for the mobile ecosystem. *arXiv preprint arXiv:2404.10518*, 2024.

[26] Jiangning Zhang, Xiangtai Li, Jian Li, Liang Liu, Zhucun Xue, Boshen Zhang, Zhengkai Jiang, Tianxin Huang, Yabiao Wang, and Chengjie Wang. Rethinking mobile block for efficient attention-based models. In *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 1389–1400. IEEE Computer Society, 2023.

[27] Sanghyun Woo, Shoubhik Debnath, Ronghang Hu, Xinlei Chen, Zhuang Liu, In So Kweon, and Saining Xie. Convnext v2: Co-designing and scaling convnets with masked autoencoders. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 16133–16142, 2023.

[28] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 10012–10022, 2021.

[29] Xiaoyi Dong, Jianmin Bao, Dongdong Chen, Weiming Zhang, Nenghai Yu, Lu Yuan, Dong Chen, and Baining Guo. Cswin transformer: A general vision transformer backbone with cross-shaped windows. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 12124–12134, 2022.

[30] Ao Wang, Hui Chen, Zijia Lin, Hengjun Pu, and Guiguang Ding. Repvit: Revisiting mobile cnn from vit perspective. *arXiv preprint arXiv:2307.09283*, 2023.

[31] Xinyu Liu, Houwen Peng, Ningxin Zheng, Yuqing Yang, Han Hu, and Yixuan Yuan. Efficientvit: Memory efficient vision transformer with cascaded group attention. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14420–14430, 2023.

[32] Glenn Jocher, Ayush Chaurasia, and Jing Qiu. Ultralytics yolov8, 2023.

[33] Chien-Yao Wang, I-Hau Yeh, and Hong-Yuan Mark Liao. Yolov9: Learning what you want to learn using programmable gradient information. In *European conference on computer vision*, pages 1–21. Springer, 2024.

[34] Ao Wang, Hui Chen, Lihao Liu, Kai Chen, Zijia Lin, Jungong Han, et al. Yolov10: Real-time end-to-end object detection. *Advances in Neural Information Processing Systems*, 37:107984–108011, 2024.

[35] Glenn Jocher and Jing Qiu. Ultralytics yolo11, 2024.

[36] Yunjie Tian, Qixiang Ye, and David Doermann. Yolov12: Attention-centric real-time object detectors. *arXiv preprint arXiv:2502.12524*, 2025.

[37] Mengqi Lei, Siqi Li, Yihong Wu, Han Hu, You Zhou, Xinhu Zheng, Guiguang Ding, Shaoyi Du, Zongze Wu, and Yue Gao. Yolov13: Real-time object detection with hypergraph-enhanced adaptive visual perception. *arXiv preprint arXiv:2506.17733*, 2025.

[38] Xu Ma, Xiyang Dai, Yue Bai, Yizhou Wang, and Yun Fu. Rewrite the stars. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2025.

[39] Yifan Feng, Jiangang Huang, Shaoyi Du, Shihui Ying, Jun-Hai Yong, Yipeng Li, Guiguang Ding, Rongrong Ji, and Yue Gao. Hyper-yolo: When visual object detection meets hypergraph computation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2024.

[40] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.

[41] Benyun Zhao, Xunkuai Zhou, Guidong Yang, Junjie Wen, Jihan Zhang, Jia Dou, Guang Li, Xi Chen, and Ben M. Chen. High-resolution infrastructure defect detection dataset sourced by unmanned systems and validated with deep learning. *Automation in Construction*, 163:105405, 2024.

[42] Jan Hosang, Rodrigo Benenson, and Bernt Schiele. Learning non-maximum suppression. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, July 2017.

[43] Yue Liu, Yunjie Tian, Yuzhong Zhao, Hongtian Yu, Lingxi Xie, Yaowei Wang, Qixiang Ye, Jianbin Jiao, and Yunfan Liu. Vmamba: Visual state space model. *Advances in neural information processing*

systems, 37:103031–103063, 2024.