Point-Query Quadtree for Crowd Counting, Localization, and More

Chengxin Liu[1] Hao Lu[1] Zhiguo Cao[1*] Tongliang Liu[2]
[1] Key Laboratory of Image Processing and Intelligent Control, Ministry of Education
School of Artificial Intelligence and Automation
Huazhong University of Science and Technology, China
[2]The University of Sydney, Australia
{cx_liu, hlu, zgcao}@hust.edu.cn

Abstract

We show that crowd counting can be viewed as a decomposable point querying process. This formulation enables arbitrary points as input and jointly reasons whether the points are crowd and where they locate. The querying processing, however, raises an underlying problem on the number of necessary querying points. Too few imply underestimation; too many increase computational overhead. To address this dilemma, we introduce a decomposable structure, i.e., the point-query quadtree, and propose a new counting model, termed Point quEry Transformer (PET). PET implements decomposable point querying via data-dependent quadtree splitting, where each querying point could split into four new points when necessary, thus enabling dynamic processing of sparse and dense regions. Such a querying process yields an intuitive, universal modeling of crowd as both the input and output are interpretable and steerable. We demonstrate the applications of PET on a number of crowd-related tasks, including fully-supervised crowd counting and localization, partial annotation learning, and point annotation refinement, and also report state-of-the-art performance. For the first time, we show that a single counting model can address multiple crowd-related tasks across different learning paradigms. Code is available at https://github.com/cxliu0/PET.

1. Introduction

Crowd counting aims to estimate the number of crowd from an image. Existing approaches typically address counting by learning surrogate targets such as density maps, where the count is acquired by integrating the inferred density map. Despite being effective, they cannot provide an intuitive understanding of the crowd, *i.e.*, no instance-level information is provided, which impedes high-level crowd analysis. Instead of merely predicting a count value from an image, some approaches focus on estimating the fine-grained information of crowd by either a head bounding box [21, 26] or a single head point [3, 17, 29]. The former casts crowd counting as a head detection problem. However, the detection accuracy has no guarantee in theory due to the lack of box information. In contrast, the latter directly outputs the head points, bypassing the error-prone stage of bounding box estimation. Nevertheless, they often require post-processing [3, 17] to obtain the location of each person. As a result, congested scenes may render failures of counting or localization. In addition, prior arts typically tackle a specific counting task or a learning paradigm; each requires a customized design. This impedes their use in different applications or tasks. For example, a fully-supervised counting model often cannot well address semi-supervised crowd counting [39].

In this work, we formulate crowd counting as a decomposable point querying process. The point querying design allows a model to receive arbitrary points as input and to reason whether each point is a person and where it locates. An appealing property of this design is that it provides an intuitive and universal modeling of crowd. To be specific, the intuition lies in that each querying point physically corresponds to a person or background. The arbitrariness of querying points implies that the position and the number of input points are both steerable. Therefore, by simply adjusting the input, our formulation naturally fits different crowd-related tasks, such as fully-supervised crowd counting and localization, partial annotation learning [39], and point annotation refinement (Fig. 1c).

However, since an input image may contain an arbitrary number of crowd, it is non-trivial to predefine the number of querying points. In practice, too few points lead to underestimation, while too many points yield a large computational cost. To tackle this pitfall, we present a decomposable structure—point-query quadtree. The key advantage of the quadtree is that it allows data-dependent splitting, where

*corresponding author

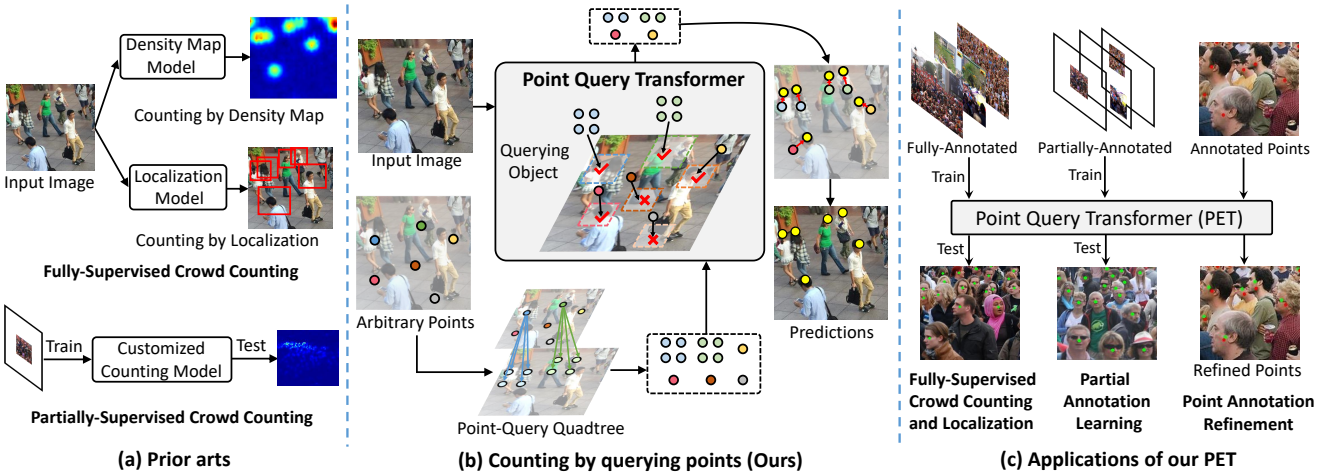


Figure 1: **Comparison between prior arts and our point-query counting paradigm**. In contrast to (a) prior arts, we consider arbitrary points as input, and reason whether each point is a person and where the person locates. We devise (b) a point-query quadtree to deal with dense crowd with adaptive tree splitting. The query design renders PET an intuitive and universal approach, enabling (c) various applications, such as fully-supervised crowd counting and localization, partial annotation learning, and point annotation refinement.

one querying point could split into several new ones when necessary, hence enabling dynamic processing of sparse and dense regions. Based on the quadtree, we instantiate a Point quEry Transformer (PET) to achieve decomposable point querying, as shown in Fig. 1b. Another key ingredient of PET is the progressive rectangle window attention, where the querying process is performed within a local window rather than the whole image in a progressive manner for efficient inference.

Extensive experiments on four crowd-counting benchmarks show that PET exhibits many appealing properties: i) *generic*: PET is applicable to several crowd-related tasks, such as fully-supervised crowd counting and localization, partial annotation learning, and point annotation refinement; ii) *effective*: PET reports state-of-the-art crowd counting and localization performance against recent approaches. In particular, it achieves a mean absolute error (MAE) of 49.34 on the ShanghaiTech PartA [42] dataset; iii) *intuitive*: PET can proceed with the point query that physically corresponds to an object or background, and outputs also interpretable points.

The contributions of this work include the following:

- We show that decomposable point querying can be a universal crowd modeling idea to potentially unify crowd-related tasks.
- We present PET, a Point Query Transformer for crowd counting, featured by the point-query quadtree and progressive rectangle window attention.

2. Related Work

We divide existing work according to whether they generate instance-level information of crowd, *i.e.*, head bounding boxes or head points. In addition, we also discuss the recent progress of transformer-based approaches.

Counting by Density Map. The majority of state-of-the-art approaches use density maps [11] as surrogate learning targets, where the count value is computed by integrating the predicted density map. These approaches advance the progress of crowd counting from various aspects, such as improving loss functions [24, 32, 34], dealing with perspective effects [1, 40, 41], and exploiting contextual information [20]. To alleviate the inconsistency between the density map and the counting value, another line of work adopts the local counting paradigm [18, 19, 37, 38], by classifying the count value of patches into discrete bins. Albeit successful, density map based approaches typically do not provide instance-level prediction. In contrast, our approach can output the location of each person with a point, enabling a more intuitive understanding of crowd.

Counting by Localization. Instead of predicting an intermediate representation like density maps, another alternative is to simultaneously estimate the count and location of crowd. Such fine-grained estimation of crowd has received substantial interest in recent years. For example, some recent approaches [13, 21, 26] cast counting as a head detection problem, predicting the bounding boxes of heads. However, the pseudo ground-truth boxes generated from weak point supervision are error-prone, especially in con-

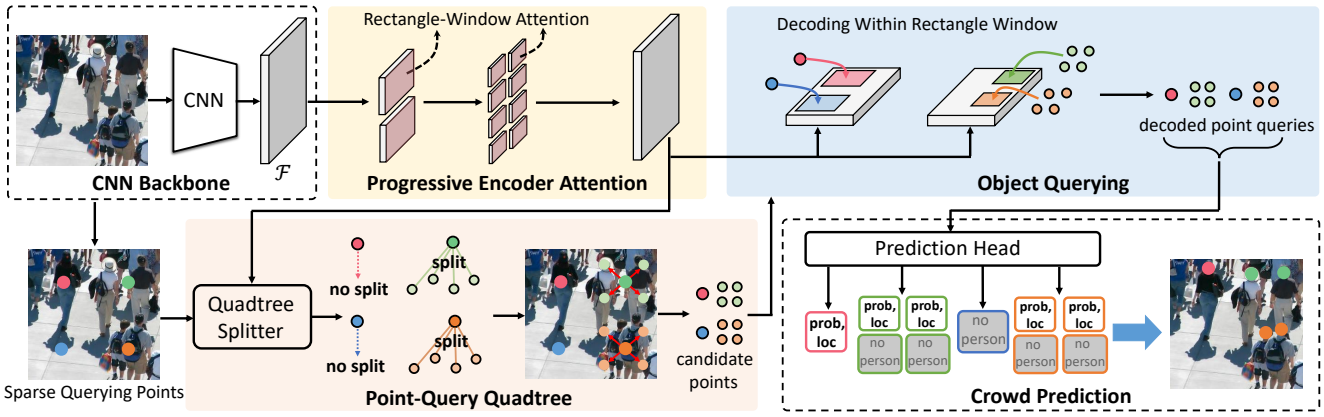


Figure 2: **Overall architecture of PET**. We first use a CNN backbone to extract the image representation $\mathcal{F}$. A transformer encoder with progressive rectangle window attention is then applied to $\mathcal{F}$ to encode context. Subsequently, the quadtree splitter receives sparse querying points and encoded features as input, outputting a point-query quadtree. The transformer decoder then decodes the point queries in parallel, where attention is computed within a local window. These point queries are finally passed through a prediction head to obtain crowd predictions, *i.e.*, 'no person' or 'a person' with its probability and localization.

gested regions. As a result, it not only hinders model training but also leads to inaccurate box predictions. Beyond bounding boxes, several approaches [3, 17, 29] directly estimate the head points. However, these approaches often need heuristic post-processing to identify individual crowd. Different from previous methods, our approach receives arbitrary points as input and predicts crowd by explicitly modeling the relation between a point and its surroundings. It streamlines the prediction process and enables applications in other crowd-related tasks.

Transformer Based Counting. The recent success of vision transformers [2, 5] has sparked their applications in various computer vision tasks. Recently, much effort [6, 14, 15, 30, 31, 36] has been devoted to deploying transformer architectures to crowd counting. Due to the strong representation capability of transformer, existing approaches mainly focus on developing a strong transformer backbone for feature extraction and incorporating it with a prediction module to estimate the count value. In contrast, we present a new formulation for crowd counting and instantiate it with a customized point query transformer.

3. Counting Crowd by Querying Points

3.1. Problem Formulation

In this work, we formulate crowd counting as a decomposable point querying process. During the querying process, sparse points could split into new points when necessary such that dense regions can be processed adaptively. In this way, counting can be achieved by querying those (split) input points.

We embody this formulation with a customized Point quEry TRansformer (PET). In PET, two ingredients are essential: i) the design of the point-query quadtree; ii) a progressive rectangle window attention mechanism. The former adaptively generates querying points to tackle dense crowd predictions, and the latter improves efficiency.

3.2. Architecture Overview

The overall architecture of PET is depicted in Fig. 2. It includes four components: a CNN backbone, an efficient encoder-decoder transformer, a point-query quadtree, and a prediction head.

Given an input image, the CNN backbone first extracts image representation, outputting the feature $\mathcal{F} \in \mathbb{R}^{h \times w \times c}$. Each spatial element in $\mathcal{F}$ is treated as a *token*, which results in $h \times w$ tokens. These tokens are then passed through a transformer encoder, implemented with progressive rectangle window attention, to encode contextual information. The rationale behind this design is efficiency, because a large number of tokens render global attention computationally expensive, especially for high-resolution images.

To obtain an instance-level understanding of the crowd, we construct a point-query quadtree to query the crowd. The point-query quadtree allows data-dependent splitting in congested regions such that dense and sparse scenes can be processed adaptively. By receiving the point-query quadtree and encoded features as input, the transformer decoder reasons the relation between point queries under the guidance of image context, and subsequently, decodes the point queries in parallel. These decoded point queries finally pass through a prediction head to acquire crowd predictions.

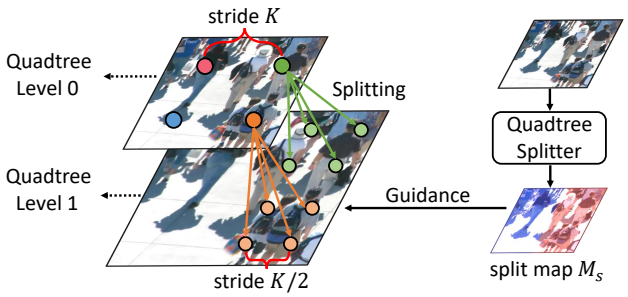


Figure 3: **Illustration of the point-query quadtree**. The split map M_s is upsampled to the original image size, where red regions require quadtree splitting.

Figure 4: **Examples of the split map**. The red regions denote the congested regions that require tree splitting.

3.3. Point-Query Quadtree

The design of the point query is vital to the success of our formulation. Considering that an input image may contain an arbitrary number of crowd, it is irrational to use a fixed number of queries as in object detection [2]. To enable scalable crowd estimation, point query should be meaningful and flexible. This yields three problems: 1) how to adapt the number of querying points to different scenes; 2) how to represent a point query; 3) how to predict crowd via point queries. We address them as follows.

Quadtree Construction. To make point query scalable to sparse and dense scenes, we propose a decomposable structure, termed point-query quadtree. The quadtree follows a sparse to dense process, *i.e.*, sparse querying points are first spanned across the image, and they are adaptively split into dense querying points in congested scenes for dense crowd prediction. A criterion is thus necessary to decide when to split sparse querying points. We consider that the splitting process should be determined by inspecting a local region, instead of relying on a single point. We therefore adopt a region-based quadtree splitter to construct the quadtree.

Quadtree Splitting. Fig. 3 illustrates the construction process of the point-query quadtree. Specifically, we first uniformly set sparse querying points on the image with a stride of K, which corresponds to the quadtree level 0. A quadtree splitter is then applied to the encoded features, outputting a split map $M_s \in \mathbb{R}^{h' \times w'}$. Each element in M_s represents the probability of a region being dense, where 1 denotes dense regions and 0 denotes sparse ones. The initial querying points in dense regions are split into dense querying points, forming the quadtree level 1. This process repeats until the maximum splitting time L is reached, which results in a $L+1$-layer quadtree. A multi-layer quadtree could be implemented by using multiple quadtree splitters to predict the split map of each layer. One may consider how many times the quadtree needs to split to tackle dense regions. In practice, we find that splitting once (L=1) is generally sufficient to deal with crowd estimation. Please refer to the supplementary for a detailed analysis.

In particular, the quadtree splitter consists of an average pooling layer and a 1×1 convolution layer followed by a `sigmoid` function, so the computational cost is negligible. We show some examples of the output of the quadtree splitter in Fig. 4, in which red regions denote congested regions requiring quadtree splitting. We observe that the quadtree splitter can distinguish the congested regions. Note that the split map is binarized using a threshold of 0.5.

Point Query Representation. Given a querying point with pixel location (x, y), we need to represent it as a point query. Our intuition tells us that a point query should contain both semantic and localization information. To encode the semantics of point query, we repurpose the CNN feature. Technically, we interpolate the CNN features $\mathcal{F}$ to the original image size and sample the feature vector $\mathcal{F}_{x,y} \in \mathbb{R}^{1\times1\times c}$. For position information, we adopt a fixed spatial positional embedding following [2]. The positional embedding and $\mathcal{F}_{x,y}$ are summed to form the point query representation.

Crowd Prediction. The final step is how to predict crowd via point queries. Following the philosophy of dotted annotations (one dot per person), we represent each person by a unique point query. Specifically, we first pass the point-query quadtree through the transformer decoder to obtain the decoded representation. A prediction head is then applied to the decoded point queries, outputting a set of predicted persons $\mathcal{Q} = \{\mathbf{q}_i\}_{i=1}^{N}$. Note that $\mathbf{q}_i$ consists of the classification probability $c_i \in [0, 1]$ and the normalized pixel location $\mathbf{p}_i = (x_i + \Delta x_i, y_i + \Delta y_i)$, where (x_i, y_i) is the pixel location of a point query and $(\Delta x_i, \Delta y_i)$ is the predicted offsets. The prediction head is composed of MLP layers with ReLU activation.

We remark that the prediction head receives a varied number of queries for different images, as the queries are generated on-the-fly by the point-query quadtree. This ensures that sparse and dense point queries only operate on corresponding regions, avoiding unnecessary computation.

3.4. Progressive Querying in Rectangle Window

Here we delineate the design of our transformer encoder and decoder. In general, we perform object querying in a progressive manner within rectangle window.

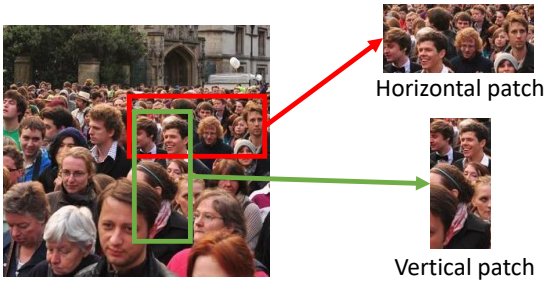


Figure 5: **The motivation of using horizontal window.** The horizontal patch often contains more people than the vertical one due to perspective prior.

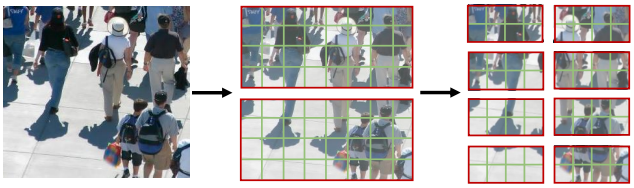


Figure 6: **Illustration of progressive encoder attention**.

Progressive Encoder Attention. To encode crowd information of different scales, we adopt a progressive attention mechanism. The idea is that the transformer encoder first inspects a sufficiently large region and then focuses on a small one. Considering that the perspective change often occurs in crowd images, we compute attention within a rectangle window. More specifically, we design a horizontal window since it usually contains more people than the vertical one (Fig. 5).

The idea of the progressive encoder attention is illustrated in Fig. 6. Given an input image, we first compute attention within a relatively large rectangle window in the first few encoder layers. Each window is with a size of $s_e \times r_e s_e$, where s_e is the height of the rectangle window, and r_e is the aspect ratio. Then, a small rectangle window (with a size of $\frac{1}{2}s_e \times \frac{1}{2}r_e s_e$) is adopted to perform attention in the subsequent encoder layers. The encoder attention is computed as:

$$\begin{aligned} \hat{\mathbf{x}}^l &= \texttt{LN}(\texttt{RectWin-SA}(\mathbf{x}^{l-1}) + \mathbf{x}^{l-1}), \\ \mathbf{x}^l &= \texttt{LN}(\texttt{FFN}(\hat{\mathbf{x}}^l) + \hat{\mathbf{x}}^l), \end{aligned} \tag{1}$$

where $\mathbf{x}^{l-1}$ and $\mathbf{x}^l$ are the output features of the encoder layer $l-1$ and the layer l, respectively. Note that $\mathbf{x}^0$ is initialized with $\mathcal{F}$. `RectWin-SA`, `FFN`, and `LN` denote rectangle window self-attention, feed-forward network, and layer normalization, respectively. Benefiting from the design of progressive rectangle window attention, we can achieve efficient computation with linear complexity, which is particularly helpful when processing high-resolution images. Detailed architecture of the encoder can be found in the supplementary.

Decoding Within Rectangle Window. The transformer decoder receives the point-query quadtree and encoded image features as input, outputting decoded point queries. It reasons crowd by modeling the relation between point queries under the guidance of image context. Intuitively, we decide whether a point query is a person based on its surrounding point queries and context. We therefore propose to compute decoder attention within local windows. Considering the hierarchy structure of quadtree, we also adopt progressive attention when decoding point queries.

Technically, the attention of sparse point queries is computed in a relatively large rectangle window (with a size of $\frac{1}{2}s_e \times \frac{1}{2}r_e s_e$), and the window size of dense point queries is reduced to $\frac{1}{4}s_e \times \frac{1}{4}r_e s_e$. In particular, the decoder attention is computed as:

$$\begin{aligned} \hat{\mathbf{z}}^l &= \texttt{LN}(\texttt{RectWin-SA}(\mathbf{z}^{l-1}) + \mathbf{z}^{l-1}), \\ \hat{\mathbf{z}}^l &= \texttt{LN}(\texttt{RectWin-CA}(\hat{\mathbf{z}}^l, \mathbf{x}^N) + \hat{\mathbf{z}}^l), \\ \mathbf{z}^l &= \texttt{LN}(\texttt{FFN}(\hat{\mathbf{z}}^l) + \hat{\mathbf{z}}^l), \end{aligned} \tag{2}$$

where $\mathbf{x}^N$ is the final output of the transformer encoder, $\mathbf{z}^{l-1}$ and $\mathbf{z}^l$ are the output features of the decoder layer $l-1$ and the layer l, respectively. Note that $\mathbf{z}^0$ is initialized with the representation of point queries. `RectWin-SA` and `RectWin-CA` denote the rectangle window self-attention and the rectangle window cross-attention, respectively.

3.5. Network Optimization

Training. The output of PET is a set of candidate persons $\mathcal{Q} = \{\mathbf{q}_i\}_{i=1}^N$. We optimize the network using bipartite matching [2] between the predictions and ground truth points $\mathcal{Y} = \{\mathbf{y}_i\}_{i=1}^M$. The loss is computed as:

$$\ell_{pq} = \frac{1}{N}\sum_{i=1}^{N} \ell_{cls}(c_i, c_i^*) + \lambda_1 \frac{1}{M}\sum_{i=1}^{M} \ell_{loc}(\mathbf{p}_{\sigma(i)}, \mathbf{y}_i), \tag{3}$$

where i is the index of a point query, c_i is the predicted classification probability, λ_1 is a hyperparameter, and $\sigma(i)$ denotes the index of point query that is matched with the ground-truth point $\mathbf{y}_i$. σ is obtained by bipartite matching [2]. The classification label c_i^* equals to 1 only if the point query is matched with a ground-truth point, and 0 otherwise. For the classification loss ℓ_{cls} and localization loss ℓ_{loc}, we employ cross entropy loss and smooth ℓ_1 loss [7]. In addition, the quadtree splitter is supervised by:

$$\ell_{split} = \mathbb{1}(\texttt{dense})(1 - \max(M_s)) + \min(M_s), \tag{4}$$

where M_s is the split map, and $\mathbb{1}(\texttt{dense})$ equals 1 if the input image has dense regions, otherwise 0. We consider an image has dense regions if its crowd density is high. Recall that each element in M_s represents the probability of a region being dense. Eq. (4) functions as a minimum supervision for the quadtree splitter to discriminate dense and

Table 1: Quantitative comparison of crowd counting results on the ShanghaiTech [42], UCF-QNRF [9], and JHU-Crowd++ [28] datasets. The best performance is in **boldface**, and the second best is underlined.

Method	Venue	Localization	SH PartA		SH PartB		UCF-QNRF		JHU-Crowd++	
			MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE
CSRNet [12]	CVPR'18	✗	68.2	115.0	10.6	16.0	-	-	85.9	309.2
CAN [20]	CVPR'19	✗	62.3	100.0	7.8	12.2	107.0	183.0	100.1	314.0
BL+ [24]	ICCV'19	✗	62.8	101.8	7.7	12.7	88.7	154.8	75.0	299.9
ASNet [10]	CVPR'20	✗	57.78	90.13	-	-	91.59	159.71	-	-
DM-Count [34]	NeurIPS'20	✗	59.7	95.7	7.4	11.8	85.6	148.3	-	-
NoisyCC [32]	NeurIPS'20	✗	61.9	99.6	7.4	11.3	85.8	150.6	67.7	258.5
SDA+DM [23]	ICCV'21	✗	55.0	92.7	-	-	80.7	146.3	59.3	248.9
GauNet+CSRNet [4]	CVPR'22	✗	61.2	97.8	7.6	12.7	84.2	152.4	69.4	262.4
GL [33]	CVPR'21	✓	61.3	95.4	7.3	11.7	84.3	147.5	59.9	259.5
P2PNet [29]	ICCV'21	✓	52.74	85.06	6.25	9.9	85.32	154.5	-	-
CLTR [14]	ECCV'22	✓	56.9	95.2	6.5	10.6	85.8	**141.3**	59.5	240.6
PET - Ours	-	✓	**49.34**	**78.77**	**6.19**	**9.69**	**79.53**	144.32	**58.5**	**238.0**

sparse regions. We retain the second term as sparse regions generally exist in an image. The final loss function is:

$$\ell_{total} = \ell_{pq} + \lambda_2 \ell_{split}\,, \tag{5}$$

where λ_2 is a weight-balancing hyperparameter.

Dual Supervision. To prevent the loss from being diluted by samples with few people, we introduce dual supervision. Given a mini-batch of samples, we divide them into sparse and dense packs according to crowd density, and separately compute ℓ_{total} on these two packs. The losses are then summed for backpropagation.

Inference. During testing, the point-query quadtree is dynamically constructed based on the split map M_s. The final predictions are obtained by thresholding the classification probability of point queries, *e.g.*, a threshold of 0.5.

4. Results and Discussion

4.1. Datasets and Implementation Details

Datasets. We evaluate our approach on four crowd counting datasets, including ShanghaiTech [42], UCF-QNRF [9], JHU-Crowd++ [28], and NWPU-Crowd [35]. Following previous work [12, 42], we use mean absolute error (MAE) and mean square error (MSE) as the evaluation metrics.

ShanghaiTech [42] has two subsets, including PartA ($300/182$ images for train/test) and PartB ($400/316$ images for train/test). UCF-QNRF [9] contains 1535 images with diverse crowd density ($1201/334$ for train/test). JHU-Crowd++ [28] is a large-scale crowd counting dataset with 4372 images ($2272/500/1600$ images for train/val/test), which covers various scenarios. NWPU-Crowd [35] is also a large-scale dataset, which contains 3109, 500, and 1500 images for training, validation, and testing, respectively.

Implementation Details. PET is optimized using the Adam optimizer [22] with the weight decay of 5×10^{-4}. The initial learning rate of 10^{-5} is set for the CNN backbone (we use VGG16 [27]), and 10^{-4} for the transformer. The point-query quadtree has a maximum depth of 2, and the initial stride of sparse point queries is set to $K = 8$. The layer number of transformer encoder and decoder is 4 and 2, respectively. Note that our point-query quadtree shares the same decoder. We set window parameters as $s_e = 16$ and $r_e = 2$. For loss coefficients, we set $\lambda_1 = 5.0$ and $\lambda_2 = 0.1$ to balance the contribution of different terms.

Regarding data augmentation, we randomly crop 256×256 patches from each image as training samples, and perform random scaling and random flipping. For datasets that contain high-resolution images, we resize the image and keep the original aspect ratio. The longer side of each image is constrained within 1536, 2048, and 2048 pixels for UCF-QNRF, JHU-Crowd++, and NWPU-Crowd, respectively.

4.2. Main Results

The point query design of PET makes it applicable to several crowd-related tasks. In this section, we demonstrate three applications, including fully-supervised crowd counting and localization, partial annotation learning, and point annotation refinement.

Comparison With State of the Art. We compare PET with state-of-the-art methods on four datasets, in the context of fully-supervised crowd counting and localization.

Table 1 reports the crowding counting results on ShanghaiTech [42], UCF-QNRF [9], and JHU-Crowd++ [28]. Our method not only achieves state-of-the-art results but also outputs localization information. In particular, PET significantly outperforms existing methods on Shang-

Table 2: Crowd counting and localization results on the NWPU-Crowd [35] test set.

(a) Crowd counting results.

Method	Venue	MAE	MSE
CSRNet [12]	CVPR'18	121.3	387.8
BL+ [24]	ICCV'19	105.4	454.2
S-DCNet [37]	ICCV'19	90.2	370.5
NoisyCC [32]	NeurIPS'20	96.9	534.2
DM-Count [34]	NeurIPS'20	88.4	388.6
P2PNet [29]	ICCV'21	77.44	362
MAN [15]	CVPR'22	<u>76.5</u>	**323.0**
PET - Ours	-	**74.4**	<u>328.5</u>

(b) Crowd localization results. **F1-m/Prec/Rec**: F-measure/precision/recall.

Method	σ_l (large threshold)			σ_s (small threshold)		
	F1-m	Prec	Rec	F1-m	Prec	Rec
Faster RCNN [25]	0.067	**0.958**	0.035	0.063	**0.894**	0.033
TinyFaces [8]	0.567	0.529	0.611	0.526	0.491	0.566
RAZNet [17]	0.599	0.666	0.543	0.517	0.576	0.470
GL [33]	0.660	<u>0.800</u>	0.562	0.587	<u>0.711</u>	0.500
D2CNet [3]	<u>0.700</u>	0.741	0.662	<u>0.632</u>	0.670	<u>0.598</u>
CLTR [14]	0.685	0.694	<u>0.676</u>	0.591	0.599	0.583
PET - Ours	**0.742**	0.752	**0.732**	**0.675**	0.684	**0.666**

Table 3: Quantitative results of partial annotation learning on the ShanghaiTech dataset. Ratio denotes the proportion of annotated region in each image. F. S. stands for fully-supervised and P. A. for partial annotation.

Method	Type	Ratio	SH PartA		SH PartB	
			MAE	MSE	MAE	MSE
MCNN [42]	F. S.	100%	110.2	173.2	26.4	41.3
CSRNet [12]	F. S.	100%	68.2	115.0	10.6	16.0
BL+ [24]	F. S.	100%	62.8	101.8	7.7	12.7
Xu *et al.* [39]	P. A.	10%	<u>72.79</u>	<u>111.61</u>	<u>12.03</u>	**18.70**
PET - Ours	P. A.	10%	**60.36**	**104.13**	**10.75**	<u>19.40</u>

haiTech PartA, with an MAE of 49.34. The good performance on UCF-QNRF and JHU-Crowd++ also supports the adaptation of PET to dense scenes, because these datasets contain images with extremely dense crowd.

For the NWPU-Crowd [35] dataset, the crowd counting results in Table 2a show that PET performs favorably against state-of-the-art methods. It is worth noticing that PET outperforms existing localization-based approaches by a considerable margin, as indicated in Table 2b. This validates the localization accuracy of PET.

Results on Partial Annotation Learning. Learning with partial annotations [39] is a new problem setting in crowd counting, in which only a partial region of each image is annotated. It aims to reduce the annotation cost and leverages data captured under various scenes. PET is applicable to this task because it can infer unlabelled regions based on the annotated ones, owing to the design of the point query. In particular, we follow the setting of [39] and simply adopt a two-step training process. We first train PET with partial annotations, then we infer the annotations around annotated regions and retrain PET with a fusion of ground-truth points and inferred annotations. In contrast to [39], we do not require a specifically designed consistency criterion to constrain model training. Table 3 shows the quantitative results on the ShanghaiTech dataset. Despite the simplicity of our training process, our approach outperforms [39] by a large margin, especially on SH PartA. In addition, PET also performs favorably against fully-supervised methods. This validates the effectiveness of PET on limited data and also the universal property of point-query modeling.

Figure 7: **Examples of point annotation refinement**. Red and green points are ground-truth points and refined points, respectively. The yellow circles highlight some refined points that significantly differ from the original ground-truth points. PET can distinguish the 'noisy' ground-truth point and move them towards the center of heads.

Application on Point Annotation Refinement. Learning with noisy annotations [16, 32] is a recent emerging topic. Existing crowd counting benchmarks typically use a dot to represent a person. However, the noise may arise during the annotation process [32], leading to inaccurate head points. In this context, another interesting application of PET is point annotation refinement. In practice, we first train PET with original annotations and validate the quality of annotations by setting annotated points as queries. Fig. 7 shows some examples of the refined annotations. We observe

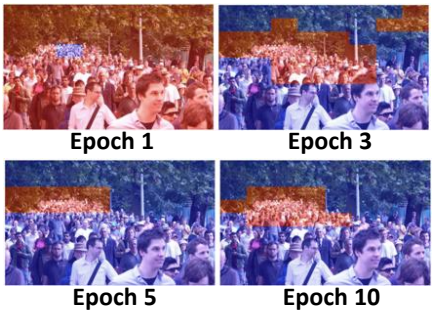


Figure 8: Split maps of epoch $1, 3, 5, 10$.

Table 4: Effect of point-query quadtree.

Configuration	SH PartA	UCF-QNRF
sparse point queries only	53.59	90.70
dense point queries only	54.16	98.14
point-query quadtree	**49.34**	**79.53**

that PET can distinguish 'noisy' annotations and calibrate these points to the center of the head. To further investigate whether the refined annotations can benefit counting performance, we retrain PET with them. Our results show that the MAE on SH PartA improves from 49.34 to 48.52. While the improvement seems marginal at first glance, we remark that this is because most annotations are reasonable in the dataset. However, the visualizations in Fig. 7 do imply the ability of PET for annotation refinement, which could make a difference in more challenging scenarios when accurate annotations are difficult to acquire.

4.3. Ablation Study

Here we justify the design choices of PET by conducting ablation studies on the ShanghaiTech PartA and UCF-QNRF datasets.

Learning Process of the Quadtree Splitter. Fig. 8 shows the outputs of the quadtree splitter during training. One can observe that the initial split map (epoch 1) is meaningless. After a few training epochs, *e.g.*, 10 epochs, the quadtree splitter can output a reasonable split map. This suggests that Eq. (4) works well in supervising the quadtree splitter.

Effect of Point-Query Quadtree. Here we verify the effectiveness of the quadtree structure. For comparison, we train PET using only sparse point queries or dense point queries. Table 4 reports the results. Although sparse point queries can achieve promising results on SH PartA, it is far from satisfactory on UCF-QNRF. In addition, the inferior results of dense point queries could be attributed to the ambiguity during bipartite matching, because a ground-truth point may correspond to several similar point queries when dealing with sparse regions. This impedes the model from discriminating valid points.

Comparatively, by adopting the point-query quadtree, we obtain notable improvements on both SH PartA ($\sim$4 MAE) and UCF-QNRF ($\sim$11 MAE). The quadtree structure ensures that sparse and dense regions can be processed by corresponding querying points, thus achieving better results.

Table 5: Effect of progressive rectangle window attention. PET is trained with different configurations of attention.

Progressive	Encoder Window	Decoder Window	MAE
✓	Rectangle	Square	52.95
✓	Square	Rectangle	51.93
✓	Square	Square	52.48
✗	Rectangle	Rectangle	52.00
✓	Rectangle	Rectangle	**49.34**

Table 6: Comparison of different attention mechanisms.

Attention Mechanism	MAE
Global Attention [2]	52.58
Deformable Attention [43]	51.0
Progressive Rectangle Window Attention (Ours)	**49.34**

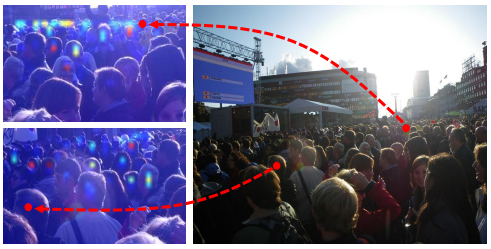
Effect of Progressive Rectangle Window Attention. To justify the effectiveness of the progressive rectangle window attention, we train with different configurations of PET as in Table 5. One can observe that: i) progressive attention works, which significantly outperforms its non-progressive counterpart; ii) applying rectangle window to the encoder and decoder achieves the best performance. We notice that replacing either encoder or decoder with square window deteriorates performance. The reason perhaps is that a rectangle window could capture more useful information about the crowd due to the perspective prior.

We also compare our attention with existing attention mechanisms. As shown in Table 6, while global attention and deformable attention obtain promising results, their performance fall behind our window attention. The inferior results of global attention could be attributed to the varied resolution of testing images, as the global-wise attention may not adapt well to arbitrary resolution. In addition, performing global attention can easily exceed the memory limit of the GPU when dealing with high-resolution images.

Efficiency of PET. Table 7 compares the model parameters and inference time of different models. Our PET owns the fewest parameters, and operates at a similar speed to existing methods. Note that the computational bottleneck lies in the backbone, instead of the querying process.

Table 7: Comparison of parameters and inference time with localization-based methods. The inference time is measured on a NVIDIA 3090 GPU with 1024×1024 input.

Method	P2PNet [29]	CLTR [14]	Ours
Parameters (M)	21.6	43.4	**20.9**
Inference Time (s)	**0.074**	0.107	0.097



(a) Encoder attention maps

(b) Decoder attention maps

Figure 9: **Visualization of encoder and decoder attention maps**. Red points in the input images are reference points.

Qualitative Results. Here we show what is learned by the transformer encoder and decoder. Fig. 9a illustrates the self-attention maps of the transformer encoder for some reference points. The encoder can capture similar crowd within a rectangle window, thus can encode valuable context information. Fig. 9b shows the decoder attention maps of two point queries. One can observe that higher attention value also occurs in similar crowd. Interestingly, the attention map of the encoder and decoder seems similar. This may attribute to the single-class nature of the crowd, *i.e.*, the model only needs to distinguish human heads.

5. Conclusion

We have shown that crowd counting can be viewed as a decomposable point querying process, which provides an intuitive and universal way of modeling crowd. Specifically, we present a Point Query Transformer, featured by a point-query quadtree structure and the progressive rectangle window attention mechanism. Extensive experiments justify that PET implements the idea of decomposable point querying and exhibits generality in several crowd-related tasks.

Albeit effective, PET still has limitations. For instance, it may suffer from missing detections when encountering large heads, because of the limited size of the rectangle window. The representation of the point query can also be improved. For future work, we plan to extend our formulation to other dense prediction tasks.

Acknowledgement This work was supported by the Natural Science Foundation of China under Grant No. U1913602, 61876211, and 62106080, and in part by the Chinese Fundamental Research Funds for the Central University under Grant 2021XXJS095.

References

[1] Shuai Bai, Zhiqun He, Yu Qiao, Hanzhe Hu, Wei Wu, and Junjie Yan. Adaptive dilated network with self-correction supervision for counting. In *CVPR*, pages 4593–4602, 2020.

[2] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. In *ECCV*, pages 213–229, 2020.

[3] Jian Cheng, Haipeng Xiong, Zhiguo Cao, and Hao Lu. Decoupled two-stage crowd counting and beyond. *IEEE TIP*, 30:2862–2875, 2021.

[4] Zhi-Qi Cheng, Qi Dai, Hong Li, Jingkuan Song, Xiao Wu, and Alexander G. Hauptmann. Rethinking spatial invariance of convolutional networks for object counting. In *CVPR*, pages 19638–19648, 2022.

[5] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *ICLR*, 2021.

[6] Junyu Gao, Maoguo Gong, and Xuelong Li. Congested crowd instance localization with dilated convolutional swin transformer. *Neurocomputing*, 513:94–103, 2022.

[7] Ross Girshick. Fast r-cnn. In *ICCV*, pages 1440–1448, 2015.

[8] Peiyun Hu and Deva Ramanan. Finding tiny faces. In *CVPR*, pages 1522–1530, 2017.

[9] Haroon Idrees, Muhmmad Tayyab, Kishan Athrey, Dong Zhang, Somaya Al-Máadeed, Nasir M. Rajpoot, and Mubarak Shah. Composition loss for counting, density map estimation and localization in dense crowds. In *ECCV*, volume 11206, pages 544–559, 2018.

[10] Xiaoheng Jiang, Li Zhang, Mingliang Xu, Tianzhu Zhang, Pei Lv, Bing Zhou, Xin Yang, and Yanwei Pang. Attention scaling for crowd counting. In *CVPR*, pages 4705–4714, 2020.

[11] Victor S. Lempitsky and Andrew Zisserman. Learning to count objects in images. In *NeurIPS*, pages 1324–1332, 2010.

[12] Yuhong Li, Xiaofan Zhang, and Deming Chen. Csrnet: Dilated convolutional neural networks for understanding the highly congested scenes. In *CVPR*, pages 1091–1100, 2018.

[13] Dongze Lian, Jing Li, Jia Zheng, Weixin Luo, and Shenghua Gao. Density map regression guided detection network for rgb-d crowd counting and localization. In *CVPR*, pages 1821–1830, 2019.

[14] Dingkang Liang, Wei Xu, and Xiang Bai. An end-to-end transformer model for crowd localization. In *ECCV*, pages 38–54, 2022.

[15] Hui Lin, Zhiheng Ma, Rongrong Ji, Yaowei Wang, and Xiaopeng Hong. Boosting crowd counting via multifaceted attention. In *CVPR*, pages 19628–19637, 2022.

[16] Chengxin Liu, Kewei Wang, Hao Lu, Zhiguo Cao, and Zhiming Zhang. Robust object detection with inaccurate bounding boxes. In *ECCV*, pages 53–69, 2022.

[17] Chenchen Liu, Xinyu Weng, and Yadong Mu. Recurrent attentive zooming for joint crowd counting and precise localization. In *CVPR*, pages 1217–1226, 2019.

[18] Liang Liu, Hao Lu, Haipeng Xiong, Ke Xian, Zhiguo Cao, and Chunhua Shen. Counting objects by blockwise classification. *IEEE Transactions on Circuits and Systems for Video Technology*, 30(10):3513–3527, 2020.

[19] Liang Liu, Hao Lu, Hongwei Zou, Haipeng Xiong, Zhiguo Cao, and Chunhua Shen. Weighing counts: Sequential crowd counting by reinforcement learning. In *ECCV*, volume 12355, pages 164–181. Springer, 2020.

[20] Weizhe Liu, Mathieu Salzmann, and Pascal Fua. Context-aware crowd counting. In *CVPR*, pages 5099–5108, 2019.

[21] Yuting Liu, Miaojing Shi, Qijun Zhao, and Xiaofang Wang. Point in, box out: Beyond counting persons in crowds. In *CVPR*, pages 6462–6471, 2019.

[22] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *ICLR*, 2019.

[23] Zhiheng Ma, Xiaopeng Hong, Xing Wei, Yunfeng Qiu, and Yihong Gong. Towards a universal model for cross-dataset crowd counting. In *ICCV*, pages 3205–3214, 2021.

[24] Zhiheng Ma, Xing Wei, Xiaopeng Hong, and Yihong Gong. Bayesian loss for crowd count estimation with point supervision. In *ICCV*, pages 6141–6150, 2019.

[25] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *IEEE TPAMI*, 39(6):1137–1149, 2017.

[26] Deepak Babu Sam, Skand Vishwanath Peri, Mukuntha Narayanan Sundararaman, Amogh Kamath, and R. Venkatesh Babu. Locate, size, and count: Accurately resolving people in dense crowds via detection. *IEEE TPAMI*, 43(8):2739–2751, 2021.

[27] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In Yoshua Bengio and Yann LeCun, editors, *ICLR*, 2015.

[28] Vishwanath Sindagi, Rajeev Yasarla, and Vishal M. M Patel. Jhu-crowd++: Large-scale crowd counting dataset and a benchmark method. *IEEE TPAMI*, pages 1–1, 2020.

[29] Qingyu Song, Changan Wang, Zhengkai Jiang, Yabiao Wang, Ying Tai, Chengjie Wang, Jilin Li, Feiyue Huang, and Yang Wu. Rethinking counting and localization in crowds: A purely point-based framework. In *ICCV*, pages 3365–3374, 2021.

[30] Guolei Sun, Yun Liu, Thomas Probst, Danda Pani Paudel, Nikola Popovic, and Luc Van Gool. Boosting crowd counting with transformers. *arXiv*, 2105.10926, 2021.

[31] Ye Tian, Xiangxiang Chu, and Hongpeng Wang. Cctrans: Simplifying and improving crowd counting with transformer. *arXiv*, 2109.14483, 2021.

[32] Jia Wan and Antoni B. Chan. Modeling noisy annotations for crowd counting. In *NeurIPS*, 2020.

[33] Jia Wan, Ziquan Liu, and Antoni B. Chan. A generalized loss function for crowd counting and localization. In *CVPR*, pages 1974–1983, 2021.

[34] Boyu Wang, Huidong Liu, Dimitris Samaras, and Minh Hoai Nguyen. Distribution matching for crowd counting. In *NeurIPS*, 2020.

[35] Qi Wang, Junyu Gao, Wei Lin, and Xuelong Li. Nwpu-crowd: A large-scale benchmark for crowd counting and localization. *IEEE TPAMI*, 43(6):2141–2149, 2021.

[36] Xing Wei, Yuanrui Kang, Jihao Yang, Yunfeng Qiu, Dahu Shi, Wenming Tan, and Yihong Gong. Scene-adaptive attention network for crowd counting. *arXiv*, 2112.15509, 2021.

[37] Haipeng Xiong, Hao Lu, Chengxin Liu, Liang Liu, Zhiguo Cao, and Chunhua Shen. From open set to closed set: Counting objects by spatial divide-and-conquer. In *ICCV*, pages 8361–8370, 2019.

[38] Haipeng Xiong, Hao Lu, Chengxin Liu, Liang Liu, Chunhua Shen, and Zhiguo Cao. From open set to closed set: Supervised spatial divide-and-conquer for object counting. *IJCV*, 131(7):1722–1740, 2023.

[39] Yanyu Xu, Ziming Zhong, Dongze Lian, Jing Li, Zhengxin Li, Xinxing Xu, and Shenghua Gao. Crowd counting with partial annotations in an image. In *ICCV*, pages 15570–15579, 2021.

[40] Zhaoyi Yan, Yuchen Yuan, Wangmeng Zuo, Xiao Tan, Yezhen Wang, Shilei Wen, and Errui Ding. Perspective-guided convolution networks for crowd counting. In *ICCV*, pages 952–961, 2019.

[41] Yifan Yang, Guorong Li, Zhe Wu, Li Su, Qingming Huang, and Nicu Sebe. Reverse perspective network for perspective-aware object counting. In *CVPR*, pages 4373–4382, 2020.

[42] Yingying Zhang, Desen Zhou, Siqin Chen, Shenghua Gao, and Yi Ma. Single-image crowd counting via multi-column convolutional neural network. In *CVPR*, pages 589–597, 2016.

[43] Xizhou Zhu, Weijie Su, Lewei Lu, Bin Li, Xiaogang Wang, and Jifeng Dai. Deformable detr: Deformable transformers for end-to-end object detection. *ICLR*, 2020.