A Review and Efficient Implementation of Scene Graph Generation Metrics

Julian Lorenz Robin Schön Katja Ludwig Rainer Lienhart
University of Augsburg, Germany
{julian.lorenz,robin.schoen,katja.ludwig,rainer.lienhart}@uni-a.de

Abstract

Scene graph generation has emerged as a prominent research field in computer vision, witnessing significant advancements in the recent years. However, despite these strides, precise and thorough definitions for the metrics used to evaluate scene graph generation models are lacking. In this paper, we address this gap in the literature by providing a review and precise definition of commonly used metrics in scene graph generation. Our comprehensive examination clarifies the underlying principles of these metrics and can serve as a reference or introduction to scene graph metrics.

Furthermore, to facilitate the usage of these metrics, we introduce a standalone Python package called SGBench *that efficiently implements all defined metrics, ensuring their accessibility to the research community. Additionally, we present a scene graph benchmarking web service, that enables researchers to compare scene graph generation methods and increase visibility of new methods in a central place.*

All of our code can be found under https://lorjul.github.io/sgbench/.

1. Introduction

Scene graph generation (SGG) aims to represent images as graphs where nodes correspond to objects in the image and edges denote relationships or interactions between the objects. The quality of generated scene graphs are pivotal in various downstream tasks, but despite the growing interest and advancements in scene graph generation models, evaluation protocols have been lacking formal definitions for the used metrics and rely on implementation details. In this paper, we address this issue by proposing comprehensive and formal definitions for scene graph generation metrics, providing a solid foundation for benchmarking and thus helping to advance research in this domain.

Many scene graph papers introduce the used metrics but don't explicitly define them. Even most survey papers on scene graph generation do not cover the metrics thoroughly. In contrast, we put much focus on a precise language to define commonly used scene graph metrics. We provide formal definitions and accompanying pseudo code to strictly describe Recall@k, Mean Recall@k, Pair Recall@k, and more. This paper thus serves as a reference and introduction to scene graph metrics. Finally, we provide evaluation results for existing panoptic scene graph methods with the discussed metrics.

In addition, we provide an efficient Python implementation of the discussed metrics. Our implementation is designed to run faster and use less disk space. On top, we use less boilerplate code and reduce the total number of lines of code from more than 1500 down to less than 700 compared to OpenPSG [24]. Our implementation is designed to be easy to understand and adaptable for future needs. All of our code is available here: https://lorjul.github.io/sgbench/.

To further advance the field of scene graph generation and improve visibility of new scene graph methods, we additionally introduce a public benchmarking web service. We aim to establish this service as a central place to compare new scene graph methods for various tasks and datasets.

To summarize, our contributions are:

1. A thorough review of commonly used metrics in scene graph generation with precise definitions.
2. A comparison of existing panoptic scene graph methods on the discussed metrics.
3. An efficient Python package called *SGBench* that is lightweight, easy to use, and supports all discussed metrics.
4. A benchmarking service where scene graph models can be evaluated and compared.

2. Related Work

2.1. Metrics

The first scene graph metric Recall@k was introduced by Lu et al. [16]. Since then many surveys have discussed the topic of scene graph generation. However, most of them do not define the used scene graph metrics. Chang et al. [4]

compare Recall@k, Mean Recall@k, and No Graph Constraint Recall@k. However, they limit the section for the metrics to a single paragraph without a thorough definition and focus more on different scene graph architectures and applications. Cheng et al. [6] only loosely define Recall@k and mention Mean Recall@k just one single time. Agarwal et al. [2] verbally define Recall@k, Mean Recall@k, and No Graph Constraint Recall@k but don't provide any strict definitions. Li et al. [14] are more precise than the previously mentioned surveys. They mathematically define Recall@k, Mean Recall@k, and No Graph Constraint Recall@k but do not dive into the details as much as we do in this paper. In addition, they define Zero-Shot Recall@k as the Recall@k for relation triplets that are not in the training set. Since this metric is just Recall@k with a different test set, we do not treat it as a separate metric in this paper.

Although most scene graph surveys loosly define the used metrics, they do not dive into the details. In this paper, we will formalize commonly used scene graph metrics and provide explicit pseudo code and thorough mathematical definitions. In addition we provide a reference implementation that was written from the ground up to be easy to use and customize.

2.2. Scene Graph Models

In the original scene graph generation task (SGG), a scene graph consists of relations between bounding boxes [13]. An extension to this approach is panoptic scene graph generation (PSGG) [24], where segmentation masks are used instead of bounding boxes. All the discussed metrics in this paper apply to both SGG and PSGG and our *SGBench* package supports both tasks. To keep this paper concise, we choose PSGG as an example to demonstrate the discussed metrics and our implementation. We argue that PSGG is a more challenging and meaningful task than SGG.

In the first paper for PSGG [24], the following methods have been ported to the panoptic task: IMP [23], MotifNet [25], GPS-Net [15], and VCTree [21]. These methods were originally designed for the SGG task but adapted for the PSGG task. PSGTR [24] and PSGFormer [24] are end-to-end scene graph architectures that are built on DETR [3] and HOTR [12]. Pair-Net [22] is an extension of PSGFormer and splits the prediction step into pair detection and then predicate classification. HiLo [27] is another one-stage scene graph method that builds on the ideas from [26]. For each relation it predicts two predicate distributions. One for low frequency predicates and one for high frequency predicates. The distributions are then merged to one single distribution to tackle the predicate imbalance issue with scene graph datasets.

For all these models, we provide conversion scripts that can be used to convert the model-specific output to our generalized format, discussed in Sec. 5.1. Future methods that build on these models can use the same conversion scripts with virtually no additional effort.

3. Terminology

To better understand the metric definitions, we define the terminology here that is used throughout this paper.

Instances An instance refers to a visual object in an image. It is identified by a segmentation mask (or bounding box) and a class label. We define M_{gt} as the set of ground truth instances in an image and M_{out} as the set of predicted instances in an image. We use the term "instances" instead of "objects" to avoid confusion with subjects/objects on relations.

Predicate A predicate is a single label that can be used to describe a relation between a subject instance and an object instance (*e.g.*, "holding", "looking at", "parked on", ...). We define P as the set that contains all possible predicate classes. The PSG dataset [24] that we use in Sec. 6.1 contains 56 elements in P.

Relation Triplet We define a relation triplet (or triplet for short) as a 3-tuple of a subject instance, a predicate label, and an object instance. Note that a triplet does not contain any scores for the predicate, the subject, or the object. If it is a ground truth triplet, we define it as $(sbj, predicate, obj) \in M_{gt} \times P \times M_{gt}$. If the triplet was returned by a scene graph model, we define it as $(sbj, predicate, obj) \in M_{out} \times P \times M_{out}$. For a triplet t, we use t_{sbj} to refer to the related subject. t_{obj} and $t_{predicate}$ are used respectively.

Scene Graph A scene graph is a graph representation that encodes interactions between visual objects in an image. It consists of a set of instances that are connected by a set of relations, defined as relation triplets.

4. Metrics

In this section, we formalize the commonly used set of metrics for scene graph generation. In contrast to existing definitions, we define them more formally and provide pseudo code for a better understanding.

4.1. Converting Scores to Triplets

Existing metrics implementations do not receive a sequence of triplets as input but directly operate on the estimated predicate confidence scores. This approach removes the flexibility that a model can decide on its own how to sort the triplets. Therefore, we decide to strictly decouple the triplet conversion step from the definition of the metrics.

Algorithm 1 General Metric Framework

Input: Output triplets X_i in descending order from a scene graph model for each image i in the dataset, ground truth triplets G_i for each image, metric function f at image-level
Output: Metric score for the whole dataset
procedure FRAMEWORK(f, G, X)
    $S \leftarrow \emptyset$
    for all images i in the dataset **do**
        $X' \leftarrow$ relevant subset of X_i
        $L \leftarrow GetMapping()$ ▷ Algorithm 2
        $X' \leftarrow ApplyMatching(L, X')$ ▷ Algorithm 3
        $s \leftarrow f(G_i, X')$ ▷ Calculate metric
        $S \leftarrow S \cup \{s\}$
    return $\overline{S}$ ▷ Average of S

The confidence scores can be converted to an ordered sequence of subject-predicate-object triplets that are used for the discussed metrics. Let s be the confidence score for a predicate p on a relation with subject sbj and object obj. Then, (sbj, p, obj) form a triplet that can be sorted by its score s. We provide scripts that perform this conversion step for all discussed scene graph methods.

4.2. General Structure

Given a sequence of triplets, all scene graph metrics follow a similar scheme, depicted in Algorithm 1.

(1) A subset of the triplets is selected. This depends on the used metric and is discussed in the respective sections. For example, most metrics have a k parameter that defines how many predicted triplets are allowed per image. **(2)** The selected triplets contain references to the predicted instances. To calculate metric scores, these references have to be replaced by references to the ground truth instances. Therefore, predicted instances are matched to ground truth instances, as discussed in Sec. 4.3. In this process, some triplets contain references to instances that cannot be matched. These triplets are removed from the selection. **(3)** The matched triplets can be compared to the ground truth triplets and a score for the image can be calculated. The exact calculation depends on the used metric and is discussed in the respective sections. **(4)** All scores for the individual images are averaged to calculate the score for the whole dataset.

Steps 2 and 4 are the same for every metric. Therefore, we define each metric by their triplet selection and score calculation in Secs. 4.4 to 4.8.

4.3. Instance Matching

The instance matching procedure consists of two parts.

First, a mapping from predicted instances to ground truth instances is generated (Algorithm 2). For each predicted instance, the ground truth instance with the same class label and the highest overlap is selected if the overlap surpasses a fixed threshold. Usually, the threshold is set to 0.5. To measure overlap between instances, IoU is used on either segmentation masks or bounding boxes depending on the type of dataset. *SGBench* supports both modes. Only a single predicted instance is allowed to be matched with a ground truth instance. Therefore, all instances that share the same matched ground truth are compared and the one with the highest overlap is selected. All other predicted instances are discarded and are matched to no ground truth at all.

Next, the mapping is used to convert a set of selected output triplets to a set of matched ground truth triplets (Algorithm 3). During this process, some triplets may not be fully matched and are thus discarded. Hence, it is not unusual that the set of selected triplets shrinks during the matching step.

Algorithm 2 Create Mapping for Instance Matching

Input: Predicted instances M_{out}, ground truth instances M_{gt}, and minimum IoU threshold t
Output: Mapping L that maps instances from M_{out} to instances in M_{gt}
procedure GETMAPPING(M_{out}, M_{gt}, t)
    Initialize mapping J with $J[x] = null \; \forall x \in M_{gt}$
    for all m in M_{out} **do**
        $M'_{gt} \leftarrow$ subset of M_{gt} that contains only instances with the same class label as m
        $x \leftarrow \arg\max_{g \in M'_{gt}} iou(g, m)$
        if $iou(x, m) > t$ **then**
            if $J[x] = null$ or $iou(x, m) > iou(x, J[x])$ **then**
                $J[x] \leftarrow m$
    $L \leftarrow J^{-1}$ ▷ Use the inverse mapping of J
    return L

Algorithm 3 Apply Instance Matching

Input: Mapping $L: M_{out} \rightarrow M_{gt}$ (Algorithm 2), selected triplets X (Sec. 4.1)
Output: Matched and filtered triplets
procedure APPLYMATCHING(L, X)
    $X' \leftarrow \emptyset$
    for all t in X_k **do**
        if $t_{sbj} \in L$ and $t_{obj} \in L$ **then**
            $t' \leftarrow (L[t_{sbj}], t_{predicate}, L[t_{obj}])$
            $X' \leftarrow X' \cup \{t'\}$
    return X'

Algorithm 4 Mean Recall for a Single Image

Input: Set of all predicate classes P, ground truth triplets G, matched triplets X_k (Algorithm 3)
Output: Mean Recall@k
procedure MEANRECALL(P, G, X_k)
    $P' \leftarrow$ subset of P that contains only predicates that exist in G
    for all predicate classes p in P **do**
        $G^{(p)} \leftarrow \{\, t \in G \mid t_{predicate} = p \,\}$
        $X^{(p)} \leftarrow \{\, t \in X_k \mid t_{predicate} = p \,\}$
    return $\frac{1}{|P'|} \sum_{p \in P'} \frac{|G^{(p)} \cap X_k^{(p)}|}{|G^{(p)}|}$

4.4. Recall@k ($R@k$)

Recall@k is the original metric for scene graph generation [16]. It measures how good a model can retrieve ground truth triplets with the top k predictions. To calculate recall, only true positives and false negatives have to be counted. Therefore, only positive ground truth annotations are required to calculate the score, which is ideal for scene graph generation, because datasets are lacking explicit negative ground truth annotation.

4.4.1 Triplet Selection

For Recall@k, a model must provide the k most confident triplets, denoted as X_k. How these triplets are selected is up to the model. In X_k, no two triplets with the same subject and object are allowed.

4.4.2 Score Definition

To calculate the per-image score, the recall between predicted triplets X_k and ground truth triplets G is returned:

$$R@k = \frac{|G \cap X_k|}{|G|} \tag{1}$$

4.5. Mean Recall@k ($mR@k$)

Mean Recall@k [5, 21] addresses the predicate imbalance in commonly used scene graph datasets by calculating a Recall@k score for each predicate individually and averaging the scores with equal weights. This ensures that rare predicates have the same influence on the final score as common predicates. The metric is depicted in Algorithm 4.

4.5.1 Triplet Selection

Mean Recall@k uses the exact same triplet selection as Recall@k (Sec. 4.4.1).

4.5.2 Score Definition

Let P' be the subset of P that contains only predicate classes that exist in the ground truth triplets G. To calculate the score for each predicate individually, the set of ground truth triplets G is split into individual sets $G^{(p)}$ for each predicate $p \in P'$. The same is done for the set of selected triplets X_k, which is split into $X_k^{(p)}$. Note that each $X_k^{(p)}$ usually contains less then k elements, because the k elements are distributed among the individual $X_k^{(p)}$.

The per-image score is calculated as shown in Eq. (2).

$$mR@k = \frac{1}{|P'|} \sum_{p \in P'} \frac{|G^{(p)} \cap X_k^{(p)}|}{|G^{(p)}|} \tag{2}$$

4.6. Pair Recall@k ($PR@k$)

Pair Recall@k [22] ignores the predicted predicate from the model and just measures how good a model can detect the existence of a relation.

4.6.1 Triplet Selection

Pair Recall@k uses the exact same triplet selection as Recall@k (Sec. 4.4.1).

4.6.2 Score Definition

Pair Recall@k ignores the predicate labels. G' denotes the set of subject-object pairs that is derived from G by stripping the predicate label from the original triplet. The same is done to the selected triplets X_k, which results in X'_k. Then, Pair Recall@k is calculated similarly to Recall@k, as defined in Eq. (3).

$$PR@k = \frac{|G' \cap X'_k|}{|G'|} \tag{3}$$

4.7. No Graph Constraint Recall@k ($ngR@k$)

No Graph Constraint Recall@k (ngR@k) [18, 25] allows any number of triplets for the same subject-object pair in the set of selected triplets as long as their predicates are different. Therefore, $ngR@k$ can evaluate second guesses from scene graph models.

In contrast to SGG, PSGG methods have not been evaluated on this metric. We fill that gap and provide a complete evaluation over all discussed PSGG methods.

4.7.1 Triplet Selection

$ngR@k$ is less restrictive than $R@k$ and allows any number of triplets that share the same subject-object combination as long as their predicates are different.

4.7.2 Score Definition

If the triplets are selected as described above, the score can be calculated like Recall@k (Sec. 4.4) but with a different X_k.

4.8. Mean No Graph Constraint Recall@k ($mNgR@k$)

$mNgR@k$ is a variant of $ngR@k$ that weights each predicate class equally. It is defined analogously to $mR@k$. The same triplet selection as in $ngR@k$ applies. The score is calculated like Mean Recall@k if X_k is replaced with the correct triplet selection process.

4.9. Choice of k

Usually, a constant absolute number is chosen for k. Common values are 20, 50, and 100. This means that on every image, the same number of triplets is selected.

In addition, *SGBench* supports values for k that are relative to the number of ground truth annotations. We denote relative k with a multiplication symbol, *e.g.*, R@$\times$10. With a relative k of 1, we can evaluate how good a model performs if it can only output as many triplets as there are ground truth annotations.

4.10. Instance Recall ($InstR$)

Scene graph model performance depends on the quality of the instance matching. Instance Recall measures how many instances are retrieved by the scene graph model. To calculate it, the mapping L from Sec. 4.3 can be used. Given a set of ground truth instances M_{gt}, Instance Recall calculates how many predicted instances M_{out} can be matched to M_{gt}.

$$InstR = \frac{|\{m \in M_{out} \mid L[m] \in M_{gt}\}|}{|M_{gt}|} \tag{4}$$

4.11. Metrics With $k = \infty$

Apart from calculating Instance Recall, we can use the predicted instances of a scene graph model and calculate what $R@k$, $mR@k$, $ngR@k$, and $mNgR@k$ scores a hypothetical perfect scene graph model could achieve given these predicted instances. We use the notation of $k = \infty$ for this case. We define $R@\infty$ as the best possible $R@k$ score given the matched instances. Algorithm 5 shows pseudo code for $R@\infty$. $mR@\infty$, $ngR@\infty$, and $mNgR@\infty$ are defined analogously.

4.12. Predicate Rank ($PRank$)

We introduce Predicate Rank ($PRank$) as a counterpart for Pair Recall. Whereas Pair Recall measures how good a scene graph model can identify subject-object pairs,

Algorithm 5 Recall@∞ for a Single Image

Input: ground truth triplets G, mapping $L\colon M_{out} \rightarrow M_{gt}$ (Algorithm 2)
Output: Recall@∞
procedure RECALL@$\infty(G, L)$
  $X' \leftarrow \emptyset$
  $J \leftarrow L^{-1}$ ▷ inverse mapping of L
  for all ground truth triplets t in G **do**
    if $J[t_{sbj}] \neq null$ and $J[t_{obj}] \neq null$ **then**
      $X' \leftarrow X' \cup \{t\}$
  return $\frac{|G \cap X'|}{|G|}$

Algorithm 6 Predicate Rank

Input: Set of all predicate classes P, ground truth triplets G, ordered sequence of all matched triplets X
Output: Predicate Rank
procedure PREDICATERANK$(P, G, X, rank)$
  for all predicates p in P **do**
    $G^{(p)} \leftarrow \{t \in G \mid t_{predicate} = p\}$
  Initialize L with $L[t] = null\ \forall t \in X$
  Initialize C with $C[t_{sbj}, t_{obj}] = 0\ \forall t \in X$
  for all predicted triplets t in X **do** ▷ Sorted
    $L[t] \leftarrow C[t_{sbj}, t_{obj}]$
    increment $C[t_{sbj}, t_{obj}]$ by 1
  $R \leftarrow \emptyset$
  for all predicates p in P **do**
    $R^{(p)} \leftarrow \emptyset$
    for all triplets t in $G^{(p)}$ **do**
      if $L[t] \neq null$ **then**
        $R^{(p)} \leftarrow R^{(p)} \cup \{L[t]\}$
    $R \leftarrow R \cup \left\{\overline{R^{(p)}}\right\}$ ▷ Add average of $R^{(p)}$
  return $\overline{R}$ ▷ Return average of R

$PRank$ measures how good a model is at choosing the correct predicate class, given a correct subject-object pair.

Within a relation, a scene graph model assigns confidence scores to each available predicate. Thus, the predicates can be sorted and the rank of the ground truth predicate can be retrieved for each relation. $PRank$ measures the average rank of the correct predicates. An average rank of 0 is best.

The related pseudo code is shown in Algorithm 6. For $PRank$, all available matched triplets are used. First, we generate a lookup table L that can map triplets to ranks. Next, the predicted ranks for all ground truth triplets are determined and averaged per predicate class. If a ground truth triplet could not be matched, the correct predicate cannot be determined. Consequently, the triplet is skipped in the calculation.

5. Implementation

Our metrics implementation *SGBench* is designed as a standalone lightweight utility and library and is available as an easy to install pip package. We have reduced the number of total dependencies to a minimum: *NumPy* [11] for efficient calculations, *Pillow* [17] to load ground truth segmentation masks that are stored as PNG images, *tifffile* [10] to load TIFF files, and *imagecodecs* [9] for LZMA or Deflate compression. There are no dependencies to large machine learning frameworks. Consequently, *SGBench* can be easily integrated into existing code bases and it is likely that newer dependency versions can be upgraded without issues.

Furthermore, we prioritized readability and ease of modification when developing *SGBench*. Our implementation uses less boilerplate code and contains less than 700 lines of code whereas the implementation from [24] contains more than 1500 lines even though we include additional metrics.

5.1. File Format

Many existing scene graph model implementations use custom file formats to store output data. To encourage interoperability between methods, we design a file format for model outputs that is independent of the underlying software stack and is easy to create. In addition, we provide utility scripts to convert existing output file formats to our new standardized format.

5.1.1 Triplets File

Triplet outputs are stored as a JSON [8] file. JSON files can be easily inspected with a text editor and are widely supported by various tools and programming languages. In many scene graph method implementations, the Pickle [20] format is used to store outputs. This format is Python-specific and not widely adopted outside the Python ecosystem. It is not self-contained if custom data structures are serialized. This means that the reader would have to have access to the same class definitions as the writer. On top, Pickle is not secure and allows arbitrary code execution. JSON on the other hand is more secure against such attacks. Additionally, it is self-contained and can be easily read from a completely different code base.

Listing 1 shows an example triplet output file. The `"version"` element is used to recognize possible future file format versions and should always be set to 1 at the moment. For each processed image, an entry in the `"images"` list is added. Each entry contains an `"id"`, which refers to the image id in the ground truth annotation file. `"seg_filename"` is a relative path to the respective TIFF file that contains the predicted segmentation masks. `"instances"` is a list that contains information about the predicted instances. Each layer in the TIFF file corresponds to an entry in the instances list. Layer 0 corresponds to entry 0, layer 1 to entry 1, and so on. `"bbox"` is the bounding box, defined in $xyxy$ format. It is used if no segmentation masks are available during evaluation. `"category"` is the 0-based class index of the instance. The `"triplets"` list is a list of *subject-object-predicate* triplets, ordered by decreasing confidence. *Subject* and *object* are indices that refer to items in the instances list. *Predicate* is the 0-based predicate class label that was inferred for the given triplet. Triplets with a higher confidence must come first. There is no limit to the length of the triplets list and *SGBench* automatically selects the correct subset of triplets based on the ordering.

Listing 1 Example triplet output file

```
{
  "version": 1,
  "images": [
    {
      "id": 123,
      "seg_filename": "seg_file.tiff",
      "instances": [
        {
          "bbox": [1, 22, 333, 44.4],
          "category": 2
        },
        // more instances ...
      ],
      "triplets": [
        [0, 3, 34],
        [2, 0, 13],
        // more triplets, ordered
        // by descending confidence ...
      ]
    },
    // more images ...
  ]
}
```

5.1.2 Segmentation Masks

For panoptic scene graph generation, additional segmentation masks are required. We use TIFF [1] images with Deflate [7] compression. Optionally, other compression methods like LZMA [19] can be used too which are slower but more effective. TIFF files allow an arbitrary number of layers and can therefore support any number of overlapping segmentation masks in contrast to PNG files. HiLo, PairNet, PSGTR, and PSGFormer all output overlapping masks. TIFF files are widely supported and have a much smaller size than storing the overlapping masks as raw NumPy arrays.

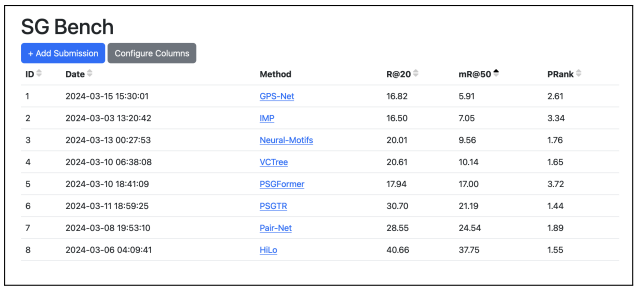

Figure 1. Screenshot of the benchmarking service interface.

5.2. Benchmarking Service

To further encourage a fair comparison between scene graph methods and improve visibility of new methods, we build a benchmarking service that runs *SGBench* under the hood. We believe that a common platform to compare recent advances in scene graph generation is beneficial to the community. The URL and server-side code can be found here: https://lorjul.github.io/sgbench/.

To upload generated scene graphs to the benchmarking server, users have to bring their model output to the format described in Sec. 5.1. The format we discussed consists of multiple files (one triplet file and several segmentation mask files). For submission, all these files have to be combined into a ZIP-archive. This archive can be uploaded to the benchmarking server.

The server then schedules the submission file for evaluation and adds the results to a publicly facing leaderboard. Figure 1 shows how the leaderboard looks like. Users can add hyperlinks to their submitted results to refer to their research. We have already added the results for all discussed methods in this paper and linked to the respective sources.

To ensure a fair competition on the leaderboard, we add an additional *approved* leaderboard where only we have upload access. We act as a central authority that periodically evaluates new published scene graph methods and uploads them to that leaderboard.

We believe that our benchmarking service can serve as a reference to compare scene graph methods on equal terms and as a place to increase visibility of published methods.

6. Experiments

6.1. Benchmark of Existing PSGG Methods

In this section, we evaluate existing scene graph methods using the described metrics and compare the results. For IMP [23], Neural Motifs [25], GPS-Net [15], VCTree [21], and HiLo [27], we use the published weights from the authors. For Pair-Net [22], no weights are provided and we train a new model using the default instructions. We convert the output files from all scene graph methods to our described format. Finally, we evaluate using all the described metrics on the converted outputs and compare the results.

Method	R@20 ↑	R@50 ↑	R@100 ↑	R@×1 ↑	R@×10 ↑
IMP	16.5	18.1	18.6	12.1	18.4
Neural-Motifs	20.0	21.7	22.0	15.1	21.8
GPS-Net	16.8	18.6	19.2	12.2	18.8
VCTree	20.6	22.1	22.5	16.0	22.3
PSGTR	30.7	35.1	35.6	19.8	35.2
PSGFormer	17.9	19.6	20.0	12.8	19.7
Pair-Net	28.5	34.3	36.8	18.4	34.9
HiLo	**40.7**	**48.7**	**51.4**	**25.2**	**49.4**

Table 1. Comparison of Recall values for different panoptic scene graph methods, evaluated on the PSG datast [24]. Higher values are better. A multiplication symbol (×) indicates a k that is relative to the number of ground truth annotations (see Sec. 4.9).

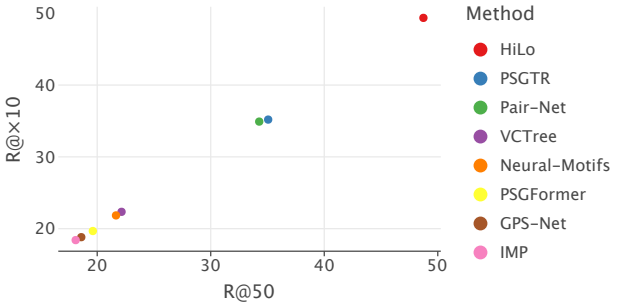

Figure 2. Absolute vs relative choice of k for $R@k$ scores. $R@50$ and $R@\times10$ are correlated with a correlation factor of about 0.9998. This shows that both choices of k are equally suited for evaluation. However, a relative k is independent of the dataset.

Method	mR@20 ↑	mR@50 ↑	mR@100 ↑	mR@×1 ↑	mR@×10 ↑
IMP	6.5	7.0	7.2	5.1	7.2
Neural-Motifs	9.1	9.6	9.7	7.8	9.6
GPS-Net	5.4	5.9	6.1	4.3	6.0
VCTree	9.7	10.1	10.2	8.1	10.2
PSGTR	18.1	21.2	21.4	10.6	21.3
PSGFormer	14.8	17.0	17.6	9.8	17.1
Pair-Net	21.5	24.5	25.6	15.9	24.9
HiLo	**29.7**	**37.8**	**41.0**	**18.8**	**39.6**

Table 2. Comparison of Mean Recall scores. Higher values are better. The column names are defined analogously to Tab. 1.

Table 1 shows Recall@k values for various k. Apart from the common $k \in \{20, 50, 100\}$, we include two relative values for k. For $R@\times10$, it is allowed to select 10 triplets per ground truth annotation. For $R@\times1$, the number of output triplets has to be the same as the number ground truth triplets. As we show in Fig. 2, similar Recall@k scores are achieved between absolute and relative k. However, the Recall@k scores with an absolute k depend on the number of ground truth annotations in the dataset. Recall@k scores with relative k are more comparable across different datasets.

Table 2 shows Mean Recall@k values. We use the same

Method	mR@50 ↑ Sec. 4.5	$mNgR@50$ ↑ Sec. 4.8	PR@50 ↑ Sec. 4.6	$PRank$ ↓ Sec. 4.12	$mR@\infty$ ↑ Sec. 4.9	$InstR$ ↑ Sec. 4.10
IMP	7.0	15.7	39.4	3.3	45.9	52.1
Neural-Motifs	9.6	23.0	40.0	1.8	45.9	52.1
GPS-Net	5.9	15.7	39.0	2.6	45.9	52.1
VCTree	10.1	23.6	40.1	1.6	45.9	52.1
PSGTR	21.2	22.8	49.2	**1.4**	66.2	65.9
PSGFormer	17.0	14.5	27.8	3.7	54.3	55.1
Pair-Net	24.5	21.9	**60.9**	1.9	**76.6**	**75.0**
HiLo	**37.8**	**37.4**	57.6	1.5	75.6	73.1

Table 3. Evaluation scores for various metrics on the PSG dataset [24]. For all metrics except $PRank$, higher values are better.

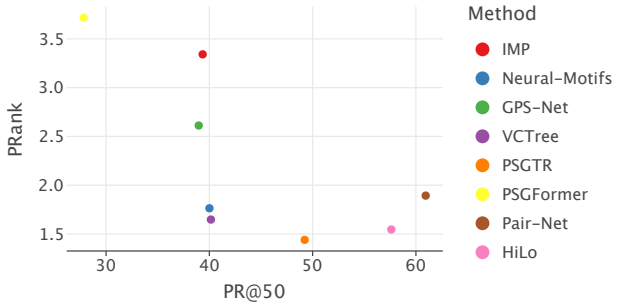

Figure 3. Pair Recall@50 ($PR@50$) compared to Predicate Rank ($PRank$). A higher $PR@50$ and a lower $PRank$ is better. A better Predicate Rank does not necessarily result in a better Pair Recall. For example PSGTR has a better $PRank$ than HiLo but a worse PR@50.

values for k as in Tab. 1. HiLo outperforms all other methods across all metrics.

The remaining metrics are shown in Tab. 3. We add $mR@50$ to the table for reference. HiLo outperforms all other methods on the main $mR@50$ and $mNgR@50$ metrics. Pair-Net is trained to improve $PR@50$ and it achieves state-of-the-art scores there as well as on $mR@\infty$ and $InstR$. However, it achieves a lower $PRank$ than HiLo which explains why Pair-Net is still inferior on the overall $mR@50$ metric. The best $PRank$ is achieved by PSGTR, but it has a comparatively low $PR@50$ which indicates why performance on $mR@50$ is not competitive. It is worth noting that the worst $PRank$ score of 3.3 by IMP is still an acceptable value, supporting the hypothesis by Wang et al. [22] that focusing on Pair Recall should be explored more in future research.

In Fig. 3, we visualize the interaction between $PR@50$ and $PRank$. Methods that perform good on one of the metrics don't necessarily improve on the other. For example, Pair-Net has the best $PR@50$ but a $PRank$ that is worse than half of the discussed scene graph methods. Pair Recall and Predicate Rank can be used as additional metrics to gain valuable insights into model performance apart from the commonly used Mean Recall@k metrics.

6.2. Comparison to Existing Implementation

All discussed scene graph methods use the same implementation to calculate the metrics [24]. In this section, we will refer to that implementation as OpenPSG, based on the corresponding code repository name.

The evaluation and metrics code of OpenPSG is tightly integrated into the rest of the code base. This makes it very difficult to use as a standalone utility in a separate environment. *SGBench* on the other hand has only four dependencies in total, is designed to be integrated into existing pipelines, and can be easily installed via pip.

SGBench supports multiple CPU cores and can thus run the evaluation much faster. While OpenPSG takes about 66 seconds to process HiLo output, *SGBench* does it in 20 seconds. Note that *SGBench* is not only faster but also calculates additional metrics that OpenPSG does not.

OpenPSG supports a slim output format via the `--submit` flag. However, this option does not allow overlapping masks which are required for PSGTR, PSGFormer, Pair-Net, and HiLo. Therefore, the only option is to use the Pickle output format. This format takes 117 GB of disk space for Pair-Net. *SGBench* requires only 761 MB, while still containing all the necessary information for evaluation.

7. Conclusion

This paper has filled a significant gap in the scene graph literature by providing a thorough review and precise definitions of existing scene graph metrics. By formalizing these metrics, we have established a solid foundation for evaluating scene graph generation models that can serve both as a reference and an introduction to scene graph generation metrics.

Furthermore, we have implemented a Python package *SGBench* that is lightweight, efficient, and easy to use. We presented our new benchmarking service and we aim to establish it as a central place to compare scene graph generation methods for various tasks.

References

[1] Adobe Systems. *TIFF (Tagged Image File Format) Specification*. Adobe Systems, 1992. 6

[2] Aniket Agarwal, Ayush Mangal, and Vipul. Visual relationship detection using scene graphs: A survey, 2020. 2

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

[4] Xiaojun Chang, Pengzhen Ren, Pengfei Xu, Zhihui Li, Xiaojiang Chen, and Alex Hauptmann. A comprehensive survey of scene graphs: Generation and application. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(1): 1–26, 2023. 1

[5] Tianshui Chen, Weihao Yu, Riquan Chen, and Liang Lin. Knowledge-embedded routing network for scene graph generation. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6156–6164, 2019. 4

[6] Jun Cheng, Lei Wang, Jiaji Wu, Xiping Hu, Gwanggil Jeon, Dacheng Tao, and Mengchu Zhou. Visual relationship detection: A survey. *IEEE Transactions on Cybernetics*, 52(8): 8453–8466, 2022. 2

[7] L. Peter Deutsch. DEFLATE Compressed Data Format Specification version 1.3. RFC 1951, 1996. 6

[8] ECMA International. *ECMA-404: The JSON Data Interchange Format*. ECMA International, 2017. 6

[9] Christoph Gohlke. cgohlke/imagecodecs: v2024.1.1, 2023. 6

[10] Christoph Gohlke. cgohlke/tifffile: v2024.2.12, 2024. 6

[11] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, 2020. 6

[12] Bumsoo Kim, Junhyun Lee, Jaewoo Kang, Eun-Sol Kim, and Hyunwoo J. Kim. HOTR: End-to-end human-object interaction detection with transformers. In *CVPR*. IEEE, 2021. 2

[13] Ranjay Krishna, Yuke Zhu, Oliver Groth, Justin Johnson, Kenji Hata, Joshua Kravitz, Stephanie Chen, Yannis Kalantidis, Li-Jia Li, David A. Shamma, Michael S. Bernstein, and Li Fei-Fei. Visual genome: Connecting language and vision using crowdsourced dense image annotations. *International Journal of Computer Vision*, 123(1):32–73, 2017. 2

[14] Hongsheng Li, Guangming Zhu, Liang Zhang, Youliang Jiang, Yixuan Dang, Haoran Hou, Peiyi Shen, Xia Zhao, Syed Afaq Ali Shah, and Mohammed Bennamoun. Scene graph generation: A comprehensive survey. *Neurocomputing*, 566:127052, 2024. 2

[15] Xin Lin, Changxing Ding, Jinquan Zeng, and Dacheng Tao. Gps-net: Graph property sensing network for scene graph generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020. 2, 7

[16] Cewu Lu, Ranjay Krishna, Michael Bernstein, and Li Fei-Fei. Visual relationship detection with language priors. In *Computer Vision – ECCV 2016*, pages 852–869, Cham, 2016. Springer International Publishing. 1, 4

[17] Andrew Murray, Hugo van Kemenade, wiredfool, Jeffrey A. Clark (Alex), Alexander Karpinsky, Ondrej Baranovič, Christoph Gohlke, Jon Dufresne, Yay295, Matthew Brett, DWesl, David Schmidt, Konstantin Kopachev, Alastair Houghton, REDxEYE, Sandro Mani, Steve Landey, Aarni Koskela, Josh Ware, vashek, Piolie, Jason Douglas, Stanislau T., David Caro, Uriel Martinez, Steve Kossouho, Riley Lahd, and Antony Lee. python-pillow/pillow: 10.2.0, 2024. 6

[18] Alejandro Newell and Jia Deng. Pixels to graphs by associative embedding. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, page 2168–2177, Red Hook, NY, USA, 2017. Curran Associates Inc. 4

[19] Igor Pavlov. LZMA SDK (software development kit), 2015. 6

[20] Antoine Pitrou. PEP 3154 - pickle protocol version 4 | peps.python.org. Technical report, Python Enhancement Proposals, 2011. 6

[21] Kaihua Tang, Hanwang Zhang, Baoyuan Wu, Wenhan Luo, and Wei Liu. Learning to compose dynamic tree structures for visual contexts. In *Conference on Computer Vision and Pattern Recognition*, 2019. 2, 4, 7

[22] Jinghao Wang, Zhengyu Wen, Xiangtai Li, Zujin Guo, Jingkang Yang, and Ziwei Liu. Pair then relation: Pair-Net for panoptic scene graph generation, 2023. 2, 4, 7, 8

[23] Danfei Xu, Yuke Zhu, Christopher Choy, and Li Fei-Fei. Scene graph generation by iterative message passing. In *Computer Vision and Pattern Recognition (CVPR)*, 2017. 2, 7

[24] Jingkang Yang, Yi Zhe Ang, Zujin Guo, Kaiyang Zhou, Wayne Zhang, and Ziwei Liu. Panoptic scene graph generation. In *ECCV*, 2022. 1, 2, 6, 7, 8

[25] Rowan Zellers, Mark Yatskar, Sam Thomson, and Yejin Choi. Neural motifs: Scene graph parsing with global context. In *Conference on Computer Vision and Pattern Recognition*, 2018. 2, 4, 7

[26] Ao Zhang, Yuan Yao, Qianyu Chen, Wei Ji, Zhiyuan Liu, Maosong Sun, and Tat-Seng Chua. Fine-grained scene graph generation with data transfer. In *ECCV*, 2022. 2

[27] Zijian Zhou, Miaojing Shi, and Holger Caesar. HiLo: Exploiting high low frequency relations for unbiased panoptic scene graph generation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 21637–21648, 2023. 2, 7