frontiers in Neuroinformatics

TECHNOLOGY AND CODE
published: 28 January 2022
doi: 10.3389/fninf.2021.767936

PyNeval: A Python Toolbox for Evaluating Neuron Reconstruction Performance

Han Zhang[1,2], *Chao Liu*[1,2], *Yifei Yu*[3], *Jianhua Dai*[4], *Ting Zhao*[5]* and *Nenggan Zheng*[1,3,4]*

[1] Qiushiq Academy for Advanced Studies (QAAS), Zhejiang University, Hangzhou, China, [2] College of Computer Science and Technology, Zhejiang University, Hangzhou, China, [3] Zhejiang Lab, Hangzhou, China, [4] Collaborative Innovation Center for Artificial Intelligence by MOE and Zhejiang Provincial Government (ZJU), Hangzhou, China, [5] Howard Hughes Medical Institute, Janelia Research Campus, Ashburn, VA, United States

Quality assessment of tree-like structures obtained from a neuron reconstruction algorithm is necessary for evaluating the performance of the algorithm. The lack of user-friendly software for calculating common metrics motivated us to develop a Python toolbox called PyNeval, which is the first open-source toolbox designed to evaluate reconstruction results conveniently as far as we know. The toolbox supports popular metrics in two major categories, geometrical metrics and topological metrics, with an easy way to configure custom parameters for each metric. We tested the toolbox on both synthetic data and real data to show its reliability and robustness. As a demonstration of the toolbox in real applications, we used the toolbox to improve the performance of a tracing algorithm successfully by integrating it into an optimization procedure.

Keywords: PyNeval, metric, quantitative analysis, neuron tracing, neuron reconstruction, toolbox

OPEN ACCESS

Edited by:
Andrew P. Davison,
UMR9197 Institut des Neurosciences Paris Saclay (Neuro-PSI), France

Reviewed by:
Hua Han,
Institute of Automation, Chinese Academy of Sciences (CAS), China
John McAllister,
Queen's University Belfast, United Kingdom

***Correspondence:**
Ting Zhao
zhaot@janelia.hhmi.org
Nenggan Zheng
zng@zju.edu.cn

Received: *31 August 2021*
Accepted: *27 December 2021*
Published: *28 January 2022*

Citation:
Zhang H, Liu C, Yu Y, Dai J, Zhao T and Zheng N (2022) PyNeval: A Python Toolbox for Evaluating Neuron Reconstruction Performance. Front. Neuroinform. 15:767936. doi: 10.3389/fninf.2021.767936

1. INTRODUCTION

Reconstructing tree structures of labeled neurons in light microscope images is a critical step for neuroscientists to study neural circuits (Parekh and Ascoli, 2013; Peng et al., 2015). Researchers have longed for automating this process of neuron reconstruction, also called neuron tracing, to overcome the bottleneck of manual annotation or proofreading (Gillette et al., 2011b; Peng et al., 2011). Despite decades of efforts (Halavi et al., 2012; Acciai et al., 2016), however, there is still no computer algorithm that can be as reliable as human labor. Besides being a complex computer vision problem itself, neuron tracing has baffled developers on how an algorithm should be evaluated. Unlike many image segmentation problems, neuron tracing has no universally accepted metric to measure its performance. In fact, it is infeasible to design one metric for all applications, which have different tolerance to different types of reconstruction errors. The real problem here is a lack of easy access to evaluation metrics. As a result, researchers have to implement a metric by themselves or compromise on metric properness for convenience. This has caused two issues in the literature. First, performance evaluation was often limited to one or two metrics that were not sufficient to offer comprehensive comparisons. Second, the metrics applied were ambiguous in general without open implementations, causing potential inconsistency and low reproducibility.

This problem can be addressed by open-source user-friendly software that allows evaluating neuron reconstruction qualities in various ways. Such software should cover the two major categories of reconstruction metrics, geometrical metrics and topological metrics. Geometrical metrics measure how well a reconstructed model overlaps with the underlying gold standard or ground truth model, while topological metrics measure the topological similarity between the two

models. Geometrical metrics are often computed by summarizing spatial matching between the two models, such as counting the number of matched nodes as done in the popular substantial spatial distance (SSD) metric (Peng et al., 2010) or measuring the length of overlapped branches in the so called length metric (Wang et al., 2011). These metrics are straightforward for telling where branches are missing or over-traced in reconstruction, but they are not suitable for evaluating topological accuracy, which is crucial in some applications such as electrophysiological simulation. For the latter situation, topological metrics such as the Digital Reconstruction of Axonal and Dendritic Morphology (DIADEM) metric (Gillette et al., 2011a), tree edit distance (Bille, 2005), and critical node (CN) metric (Feng et al., 2015) are preferred.

Hence, we introduce a Python toolbox called PyNeval, which is the first open-source toolbox designed to provide multiple choices for evaluating the qualities of reconstruction results conveniently. In specific, PyNeval is designed to have the following features:

- PyNeval has a user-friendly command-line interface for easy use and a flexible way of configuring parameters for covering a broad range of user requirements.
- PyNeval provides various evaluation methods for measuring both geometrical and topological qualities of reconstructions.
- PyNeval provides an interface for optimizing any reconstruction algorithm that converts an image into an SWC file with adjustable parameters.

In this paper, we formulate each evaluation method implemented in PyNeval under a mathematical framework if it has not been clearly defined in the literature. Our implementation follows those formulations, which give users a clear and unambiguous picture of what PyNeval computes. We apply PyNeval to randomly perturbed data to show that PyNeval can produce reliable evaluation scores from different metrics. The difference among the metrics can be seen in their results of manually-designed special cases. Besides comparing different tracing algorithms, PyNeval can be used to optimize any reconstruction algorithm with tunable parameters, as demonstrated in our experiment on mouse brain data acquired by fMOST (Gong et al., 2016).

2. METHOD

2.1. SWC Format

The PyNeval toolbox is designed based on the SWC format (Cannon et al., 1998), the common format of neuron reconstruction results. The format represents the shape of a neuron in a tree structure that consists of a set of hierarchically organized nodes (Feng et al., 2015):

$$T = \{\mathbf{n}_i = (\mathbf{x}_i, r_i, \mathbf{n}_j) \mid i = 1, ..., N_T, \mathbf{n}_j \in T \cup \mathbf{n}_0, i \neq j, \\ \mathbf{x}_i \in \mathbb{R}^3, r_i \in \mathbb{R}\} \quad (1)$$

where $N_T = |T|$ is the number of nodes of T, the ith node $\mathbf{n}_i$ is a sphere centering at $\mathbf{x}_i = (x_i, y_i, z_i)$ with radius r_i, and $\mathbf{n}_0$ is a virtual node. In this definition, $\mathbf{n}_j$ is called the parent of $\mathbf{n}_i$, and a node with a virtual node as its parent is called a root node. For convenience, we also define the following functions:

- Parent of a node: $\rho : (\mathbf{x}_i, r_i, \mathbf{n}_j) \in T \mapsto \mathbf{n}_j \in T \cup \mathbf{n}_0$
- Position of a node: $\mathbf{x} : (\mathbf{x}_i, r_i, \mathbf{n}_j) \in T \mapsto \mathbf{x}_i \in \mathbb{R}^3$
- Radius of a node: $r : (\mathbf{x}_i, r_i, \mathbf{n}_j) \in T \mapsto r_i \in \mathbb{R}$

The edge set of the model T is defined as

$$E(T) = \{\mathbf{e}_i \mid \mathbf{e}_i = (\mathbf{n}_i, \rho(\mathbf{n}_i)), \mathbf{n}_i \in T\} \quad (2)$$

One important constraint on the SWC model T is that the graph $G = (T \cup \mathbf{n_0}, E(T))$ has no loop, which means that it is a tree.

2.2. Software Design

Assuming that the reconstruction results are in the SWC format, PyNeval takes a gold standard SWC file as well as one or more testing SWC files and outputs the quality scores for each testing SWC. Since PyNeval supports multiple metrics, it should also allow the user to specify metric options. As a consequence, input SWC files and metric options form the essential parameters of the main PyNeval command. While this provides a straightforward interface for an application, it is not flexible enough to adapt to more subtle user requirements such as setting specific parameters for a certain metric or checking evaluation details. Therefore, PyNeval has a flexible but friendly way of accepting optional parameters, allowing the user to specify these parameters without having to check extensive documents. PyNeval can output carefully formatted results to the screen for easy reading or save the results with more details to a file for further analysis, depending on the user's choice of the output parameters. For example, the `-detail` option can be used to produce an SWC file that labels each node in the test structure with a specific type to indicate what kind of error is associated with that node. The overall architecture of PyNeval is shown in **Figure 1**.

2.3. Metrics

PyNeval supports four commonly used metrics in both geometrical and topological categories, although it can be easily extended to more metrics. To explain the metrics implemented in PyNeval unambiguously, we use the notations listed in **Table 1**.

More specifically, some of the notations can be interpreted as follows:

- Besides always assuming that $\mathbf{e}_{ij} = (\mathbf{n}_i, \mathbf{n}_j)$ and $\mathbf{e}_i = (\mathbf{n}_i, \rho(\mathbf{n}_i))$, we also use $\mathbf{n}$ and $\mathbf{e}$ to represent a node and an edge, respectively, when there is no need to index them.
- Node interpolation

$$\mathcal{I}(\mathbf{n}, \lambda) = \begin{cases} ((1-\lambda)\mathbf{x}(\mathbf{n}) + \lambda\mathbf{x}(\rho(\mathbf{n})), (1-\lambda)r(\mathbf{n}) \\ \quad + \lambda r(\rho(\mathbf{n})), \rho(\mathbf{n}))), & 0 \leq \lambda < 1 \\ \rho(\mathbf{n}), & \lambda = 1 \end{cases} \quad (3)$$

- Interpolation between two nodes, no matter if they are connected

$$\mathcal{I}(\mathbf{n}_i, \mathbf{n}_j, \lambda) = \begin{cases} ((1-\lambda)\mathbf{x}_i + \lambda\mathbf{x}_j, (1-\lambda)r_i \\ \quad + \lambda r_j, \mathbf{n}_j), & 0 \leq \lambda < 1 \\ \mathbf{n}_j, & \lambda = 1 \end{cases} \quad (4)$$

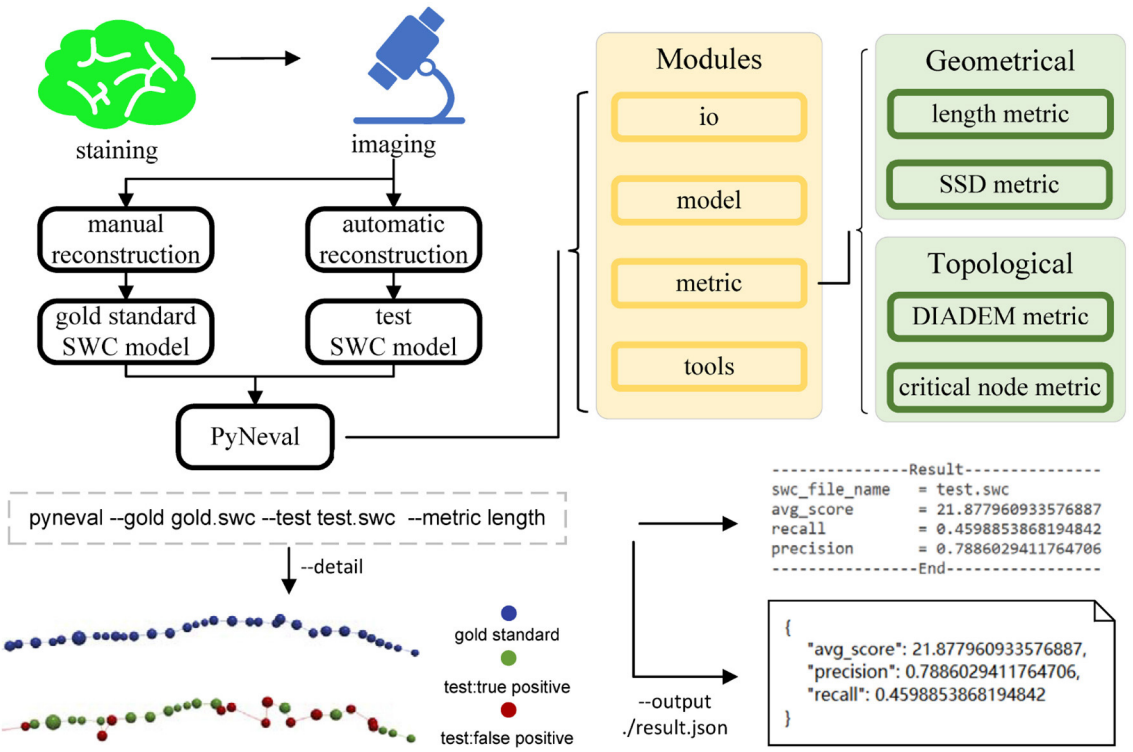

FIGURE 1 | The overview of PyNeval, which can take a gold standard model and a test model from the same image as input through a command line interface and outputs quality scores of the test model, as well as more details about reconstruction errors. Four different metrics, including the length and substantial spatial distance (SSD) metrics in the geometrical category and the DIADEM and critical node (CN) metrics in the topological category, are available in PyNeval.

TABLE 1 | Mathematical notations used for explaining the metrics.

Symbol	Meaning
T_g	Gold standard SWC model
T_t	SWC model for evaluation
$\mathbf{n}_i$	A node in a SWC model with an unique index i
$E(T)$	Set of all edges in T
$\mathbf{e}_i$	Edge from node $\mathbf{n}_i$ to node $\rho(\mathbf{n}_i)$
$d(x, y)$	Distance between two objects, which can be nodes, edges or trees
L	Length of an edge or an edge set
M	Matched node or edge set
$\mathcal{I}$	Interpolation function

- Node distances

$$d(\mathbf{n}_i, \mathbf{n}_j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2} \tag{5}$$

$$d_{xy}(\mathbf{n}_i, \mathbf{n}_j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \tag{6}$$

$$d_z(\mathbf{n}_i, \mathbf{n}_j) = |z_i - z_j| \tag{7}$$

$$d(\mathbf{n}, \mathbf{e}_i) = \min_{\lambda} d(\mathbf{n}, \mathcal{I}(\mathbf{n}_i, \lambda)) \tag{8}$$

$$d(\mathbf{n}, T) = \min_{\mathbf{e} \in E(T)} d(\mathbf{n}, \mathbf{e}) \tag{9}$$

- Node length

$$L(\mathbf{n}) = \begin{cases} 0, & \mathbf{n} \text{ is a root node} \\ d(\mathbf{n}, \rho(\mathbf{n})), & \text{otherwise} \end{cases} \tag{10}$$

- Edge lengths

$$L(\mathbf{e}_{ij}) = d(\mathbf{n}_i, \mathbf{n}_j) \tag{11}$$

$$L(E(T)) = \sum_{\mathbf{e} \in E(T)} L(\mathbf{e}) \tag{12}$$

- Tree length

$$L(T) = L(E(T)) \tag{13}$$

2.3.1. Length Metric

It is natural to evaluate the quality of a reconstruction T_t by measuring how well its branches overlap with the gold standard model T_g. This can be computed by matching edges between T_t

and T_g and then summing up the lengths of the matched edges in T_t and T_g, respectively, to produce the common precision and recall metrics. Before proceeding to explain the length metric in detail, we first need to define some more notations

- A segment lying on an edge $(\mathbf{n}, \rho(\mathbf{n}))$ is

$$C(\mathbf{n}, \lambda_1, \lambda_2) = \{\mathbf{x}(\mathcal{I}(\mathbf{n}, \lambda)) | 0 \leq \lambda_1 \leq \lambda \leq \lambda_2 \leq 1\} \tag{14}$$

  and its length is $L(C(\mathbf{n}, \lambda_1, \lambda_2)) = (\lambda_2 - \lambda_1)L(\mathbf{n})$.
- The overlap ratio between two segments C_1, C_2 with respect to the edge $(\mathbf{n}, \rho(\mathbf{n}))$ is defined as $\mathcal{O}(C_1, C_2)$. Suppose that $C_1 = (\mathbf{n}_i, \alpha_1, \alpha_2)$, $C_2 = C(\mathbf{n}_j, \beta_1, \beta_2)$, the overlap ratio $\mathcal{O}(C_1, C_2)$ is

$$\mathcal{O}(C_1, C_2) = \begin{cases} \max(0, \min(\alpha_2 - \alpha_1, \\ \beta_2 - \beta_1, \alpha_2 - \beta_1, \beta_2 - \alpha_1)), & i = j \\ 0, & i \neq j \end{cases} \tag{15}$$

- A simple path between two points on a tree is

$$\mathcal{P}((\mathbf{n}_s, \lambda_s), (\mathbf{n}_t, \lambda_t)) = \begin{cases} \{C(\mathbf{n}_s, \min(\lambda_s, \lambda_t), \max(\lambda_s, \lambda_t))\}, & s = t \\ \{C(\mathbf{n}_{i_k}, \alpha_k, \beta_k) | k = 1 \cdots K\}, & s \neq t \end{cases} \tag{16}$$

  where $i_1 = s$, $i_K = t$, $\mathbf{n}_{i_{k-1}} = \rho(\mathbf{n}_{i_k})$ or $\mathbf{n}_{i_k} = \rho(\mathbf{n}_{i_{k-1}})$, K is the number of edges on the path and

$$(\alpha_k, \beta_k) = \begin{cases} (0, \lambda_s), & k = 1 \text{ and } \mathbf{n}_s = \rho(\mathbf{n}_{i_2}) \\ (\lambda_s, 1), & k = 1 \text{ and } \rho(\mathbf{n}_s) = \mathbf{n}_{i_2} \\ (0, \lambda_t), & k = K \text{ and } \mathbf{n}_t = \rho(\mathbf{n}_{i_{K-1}}) \\ (\lambda_t, 1), & k = K \text{ and } \rho(\mathbf{n}_t) = \mathbf{n}_{i_{K-1}} \\ (0, 1), & \text{otherwise} \end{cases} \tag{17}$$

In our implementation, we construct the matched edge set between T_t and T_g as demonstrated in **Algorithm 1**.

2.3.2. SSD Metric

The SSD metric (Peng et al., 2010) can be viewed as a variant of the length metric in terms of what it tries to measure. Instead of matching edges directly, however, SSD counts how many nodes are matched without excluding duplicated matches. Besides, SSD provides an additional metric to measure how far the unmatched nodes are away from the counterpart model. One extra step of SSD metric is resampling each branch of T_t and T_g uniformly to reach a sufficient density ε_{sp}

$$\mathcal{R}(T) = \{\mathbf{n}_k^{(i)} | \mathbf{n}_i \in T, k = 0, 1, \ldots, K_i\} \tag{18}$$

where $\mathbf{n}_k^{(i)} = \mathcal{I}(\mathbf{n}_i, \mathbf{n}_{k+1}^{(i)}, \frac{k}{k+1})$, $\mathbf{n}_{K_i}^{(i)} = \mathbf{n}_0^{(j)}$, $\rho(\mathbf{n}_i) = \mathbf{n}_j$, $K_i \varepsilon_{sp} \leq L(\mathbf{e}_{ij})$, and $(K_i + 1)\varepsilon_{sp} > L(\mathbf{e}_{ij})$.

After that, like computing the length metric, the SSD metric can be obtained by constructing the matched node set $M_{\mathbf{n}}$ between two SWC models T_g and T_t shown in **Algorithm 2**.

2.3.3. CN Metric

The CN metric measures how many CNs are reconstructed correctly. A critical node is either a branching or terminal node, which determines the topology of an SWC model. Mathematically, the set of the CNs of an SWC model T is defined as

$$\mathcal{K}(T) = \{\mathbf{n} | \mathbf{n} \in T, D_T(\mathbf{n}) \neq 2\} \tag{19}$$

where $D_T(\mathbf{n})$ is the degree of node $\mathbf{n}$ in the tree T.

Algorithm 1: Length metric.

Input: $T_g, T_t, \epsilon_l \in \mathbb{R}^+, \epsilon_o \in \mathbb{R}^+, \epsilon_d \in \mathbb{R}^+$
Output: precision, recall

$M_t \leftarrow \emptyset, M_g \leftarrow \emptyset$
for $\mathbf{e}_{ij}$ in $E(T_t)$ **do**
  $\mathcal{I}(\mathbf{n}'_{g_1}, \lambda_1) \leftarrow \arg\min_{\mathbf{n}' \in \cup_{\mathbf{n} \in T_g} \{\mathcal{I}(\mathbf{n}, \lambda) | 0 \leq \lambda \leq 1\}} d(\mathbf{n}_i, \mathbf{n}')$
  $\mathcal{I}(\mathbf{n}'_{g_2}, \lambda_2) \leftarrow \arg\min_{\mathbf{n}' \in \cup_{\mathbf{n} \in T_g} \{\mathcal{I}(\mathbf{n}, \lambda) | 0 \leq \lambda \leq 1\}} d(\mathbf{n}_j, \mathbf{n}')$
  if $\max(d(\mathbf{n}_i, \mathcal{I}(\mathbf{n}'_{g_1}, \lambda_1)), d(\mathbf{n}_j, \mathcal{I}(\mathbf{n}'_{g_2}, \lambda_2))) < \epsilon_d$ **then**
    $\mathcal{P}((\mathbf{n}'_{g_1}, \lambda_1), (\mathbf{n}'_{g_2}, \lambda_2))$ is the simple path between $\mathcal{I}(\mathbf{n}'_{g_1}, \lambda_1)$ and $\mathcal{I}(\mathbf{n}'_{g_2}, \lambda_2)$
    if $\frac{|L(\mathbf{e}_{ij}) - L(\mathcal{P}((\mathbf{n}'_{g_1}, \lambda_1), (\mathbf{n}'_{g_2}, \lambda_2)))|}{L(\mathbf{e}_{ij})} < \varepsilon_l$ and $\max_{C_1 \in \mathcal{P}((\mathbf{n}'_{g_1}, \lambda_1), (\mathbf{n}'_{g_2}, \lambda_2)), C_2 \in M_g} \mathcal{O}(C_1, C_2) < \varepsilon_o$ **then**
      $M_t \leftarrow M_t \cup \{\mathbf{e}_{ij}\}$
      $M_g \leftarrow M_g \cup \mathcal{P}((\mathbf{n}'_{g_1}, \lambda_1), (\mathbf{n}'_{g_2}, \lambda_2))$
    end if
  end if
end for
precision $\leftarrow \frac{L(M_t)}{L(T_t)}$
recall $\leftarrow \frac{L(M_g)}{L(T_g)}$
return precision, recall

Algorithm 2: SSD metric.

Input: $\mathcal{R}(T_g), \mathcal{R}(T_t), \epsilon_{sp} \in \mathbb{R}^+, \epsilon_{ssd} \in \mathbb{R}^+$
Output: precision, recall, SSD cost

$M_n(T_g, T_t) \leftarrow \emptyset, M_n(T_t, T_g) \leftarrow \emptyset$
for $\mathbf{n}_i$ in $\mathcal{R}(T_g)$ **do**
  if $\min_{\mathbf{n}_j \in \mathcal{R}(T_t)} d(\mathbf{n}_i, \mathbf{n}_j) < \varepsilon_{ssd}$ **then**
    $M_n(T_g, T_t) \leftarrow M_n(T_g, T_t) \cup \{\mathbf{n}_i\}$
  end if
end for

for $\mathbf{n}_i$ in $\mathcal{R}(T_t)$ **do**
  if $\min_{\mathbf{n}_j \in \mathcal{R}(T_g)} d(\mathbf{n}_i, \mathbf{n}_j) < \varepsilon_{ssd}$ **then**
    $M_n(T_t, T_g) \leftarrow M_n(T_t, T_g) \cup \{\mathbf{n}_i\}$
  end if
end for

precision $\leftarrow \frac{|M_n(\mathcal{R}(T_t), \mathcal{R}(T_g))|}{|\mathcal{R}(T_t)|}$
recall $\leftarrow \frac{|M_n(\mathcal{R}(T_g), \mathcal{R}(T_t))|}{|\mathcal{R}(T_g)|}$
SSD cost $\leftarrow \frac{SSD(\mathcal{R}(T_t), \mathcal{R}(T_g)) + SSD(\mathcal{R}(T_g), \mathcal{R}(T_t))}{2}$
return precision, recall, SSD cost

Algorithm 3: Critical node metric.

Input: $\mathcal{K}(T_g), \mathcal{K}(T_t), \epsilon_{br} \in \mathbb{R}^+$
Output: precision, recall
$V_b \leftarrow \mathcal{K}(T_t) \cup \mathcal{K}(T_g)$
$E_b \leftarrow \{(\mathbf{n}^{(t)}, \mathbf{n}^{(g)}) | \mathbf{n}^{(t)} \in \mathcal{K}(T_t), \mathbf{n}^{(g)} \in \mathcal{K}(T_g), d(\mathbf{n}^{(t)}, \mathbf{n}^{(g)}) < \varepsilon_{br}\}$
$G_b \leftarrow (V_b, E_b)$
$M_b^* \leftarrow \arg\max_{M_b} |M_b|$ #M_b is a matching in G_b, i.e., M_b is a subgraph of G_b and all of its nodes have degree 1.
precision $\leftarrow \frac{|M_b^*|}{|\mathcal{K}(T_t)|}$
recall $\leftarrow \frac{|M_b^*|}{|\mathcal{K}(T_g)|}$
return precision, recall

Algorithm 4: Diadem metric.

Input: $\mathcal{K}(T_g), \mathcal{K}(T_t), \epsilon_{xy} \in \mathbb{R}^+, \epsilon_z \in \mathbb{R}^+, , \epsilon_{ld} \in \mathbb{R}^+$
Output: DIADEM score
for $\mathbf{n}_i \in \mathcal{K}(T_g)$ **do**
  for $\mathbf{n}_j \in \mathcal{K}(T_t)$ **do**
    if $d_{xy}(\mathbf{n}_i, \mathbf{n}_j) < \epsilon_{xy}$ and $d_z(\mathbf{n}_i, \mathbf{n}_j) < \epsilon_z$ **then**
      # search for $\alpha(\mathbf{n})$, the ancestor of $\mathbf{n}$ on the path between $\mathbf{n}$ and its root $\mathbf{n}_0$.
      # $\mathbf{n}_0^{(g)}$, $\mathbf{n}_0^{(t)}$ are the roots of gold and test trees respectively.
      for $\alpha(\mathbf{n}_i)$ in $\mathcal{P}((\mathbf{n}_i, 0), (\mathbf{n}_0^{(g)}, 0))$ **do**
        for $\alpha(\mathbf{n}_j)$ in $\mathcal{P}((\mathbf{n}_j, 0), (\mathbf{n}_0^{(t)}, 0))$ **do**
          if $\alpha(\mathbf{n}_i)$ matches $\alpha(\mathbf{n}_j)$ and $\frac{|L(\mathcal{P}(\mathbf{n}_i, \alpha(\mathbf{n}_i))) - L(\mathcal{P}(\mathbf{n}_j, \alpha(\mathbf{n}_j)))|}{L(\mathcal{P}(\mathbf{n}_i, \alpha(\mathbf{n}_i)))} < \epsilon_{ld}$ **then**
            $M_d \leftarrow M_d \cup \{\mathbf{n}_i\}$
          end if
        end for
      end for
    end if
  end for
end for
DIADEM score $= \frac{\sum_{\mathbf{n} \in M_d} D_{T_g}(\mathbf{n})}{\sum_{\mathbf{n} \in \mathcal{K}(T_g)} D_{T_g}(\mathbf{n})}$
return DIADEM score

With the CNs, we can compute the CN metric with **Algorithm 3**.

2.3.4. DIADEM Metric

Introduced by Gillette et al. (2011a) for the DIADEM challenge (Gillette et al., 2011b), the DIADEM metric evaluates the similarity between two models by comparing their branching structures. Like the CN metric, the DIADEM metric is also based on matching CNs in $\mathcal{K}(T_g)$ and $\mathcal{K}(T_t)$, here $\mathcal{K}(T)$ is defined in equation (19). But its matching criteria are more complicated than simply checking the distances. A brief description of the DIADEM metric is proposed as **Algorithm 4**.

TABLE 2 | Summary of neuron reconstructions from six image stacks.

ID	Number of nodes	Number of roots	Source
BN1	4,966	7	BigNeuron
BN2	852	7	BigNeuron
BN3	432	2	BigNeuron
BN4	4,251	4	BigNeuron
FM1	5,160	63	fMOST
FM2	674	9	fMOST

There are also several rounds of scanning to deal with the problem that $\mathbf{n}_j \in \mathcal{K}(T_t)$ is not the only node that meets the conditions, and labels every unmatched CN in T_g as a match if it is on a matched path. More details can be found in the reference (Gillette et al., 2011a).

2.4. Implementation

PyNeval is implemented in Python 3 (Oliphant, 2007) using several powerful open-source packages, including Numpy (Van Der Walt et al., 2011) for numerical computation, Anytree (Anytree., 2020) for handling the SWC data structure, and kdtree as well as Rtree (KDtree., 2017; Rtree., 2020) for fast search of closest edges and nodes.

3. RESULTS

3.1. Robustness Test

We applied PyNeval to randomly perturbed gold standard reconstructions to characterize each metric and evaluate the robustness of our program. The perturbed dataset is constructed by randomly moving a portion of nodes in the original reconstructions, which are gold standard SWC models from the standard BigNeuron dataset (Peng et al., 2015) as well as our custom dataset acquired from fMOST (Gong et al., 2016). As listed in **Table 2**, a total of six reconstructions with a large variety of sizes were used for the test.

A reasonable metric should produce decreasing quality scores as the perturbation ratio increases. This can be seen in the experimental results plotted in **Figure 2**, in which each curve shows the trend of a metric score along the increasing perturbation ratio. Each metric score at a perturbation ratio was averaged from 10 trials for a sequence of 11 perturbation ratios increasing by the step of 0.1 from 0 to 1. As expected, the curves are consistently similar among different models, in spite of their different morphologies. They all follow the right trend that more perturbation results in a worse score. We can also see that, topological metrics have higher variance than geometrical metrics, which is not surprising because how a perturbation affects the topology highly depends on the positions of the perturbed nodes. This suggests that when we use a topological metric to evaluate an algorithm, more samples or trials might be needed to draw a reliable conclusion.

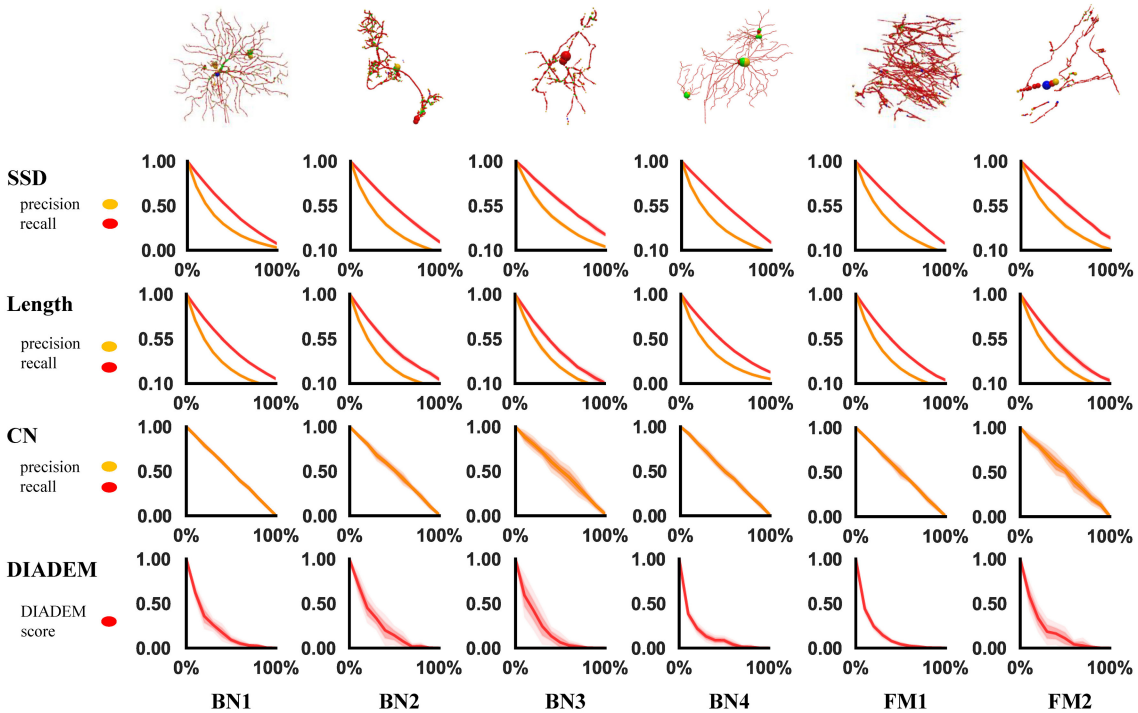

FIGURE 2 | Result of robustness test. Each row represents a metric and each column represents an swc model. In each chart, the x-axis is the perturbed proportion and the y-axis is the corresponding metric value.

3.2. Special Case Analysis

In addition to the perturbation experiment, we also tested the behaviors of the metrics on some special cases to show their differences more clearly. We constructed four special cases for geometrical metrics and the other four for topological metrics, including **Figure 3**:

- Test cases for geometrical metrics
  1. Both ends of an edge in T_g have matched nodes in T_t, but T_t has an extra node that deviates the path from the edge segment in T_g.
  2. T_g manages to find a match path in T_t, but its nodes do not match those on the same path in T_g due to the sampling rate.
  3. A straight path in T_g is reconstructed as a bifurcation in T_t by mistake.
  4. T_t distorts a relatively straight path in T_g into a zigzag path.
- Test cases for topological metrics
  1. The nodes are matched but a wrong connection in T_t changes the root to a non-CN.
  2. T_t has wrong connections, but all the CNs are still matched between T_t and T_g.
  3. The reconstruction moves a node and all its descendants to a different location.
  4. Connection mistakes in T_t break the original model into several isolated graphs.

Table 3 shows different results on the same special cases produced by the SSD and length metrics. The SSD metric tends to output higher F1 scores than the length metric does, but it is not necessarily better or worse. In some cases (**Figures 3A,B**), the SSD scores look more reasonable because their more granulated matching can capture partial matching of a path. In other cases (**Figures 3C,D**), where the errors are more complicated, however, the SSD metric can overestimate reconstruction qualities by counting duplicated matches.

The difference between the two topological metrics can be seen in **Figure 4** and **Table 4**. The CN metric fails to detect reconstruction errors in **Figures 4A,D** because the errors do not add or remove a critical node. The DIADEM metric can avoid such a problem by including path comparison. In this sense, the DIADEM metric is more comprehensive than the other three metrics in PyNeval as it actually considers both topological and geometrical features. Nevertheless, we should note that it may not correlate well with the amount of editing work needed to fix errors. For example, the test model in case 2 can be more readily fixed than case 3, despite that it has a lower DIADEM score. In other words, the DIADEM metric can be misleading when we expect an automatic method to minimize manual work.

3.3. Reconstruction Parameter Optimization Using PyNeval

Besides comparing different reconstruction algorithms, another important application of PyNeval is to optimize parameters of

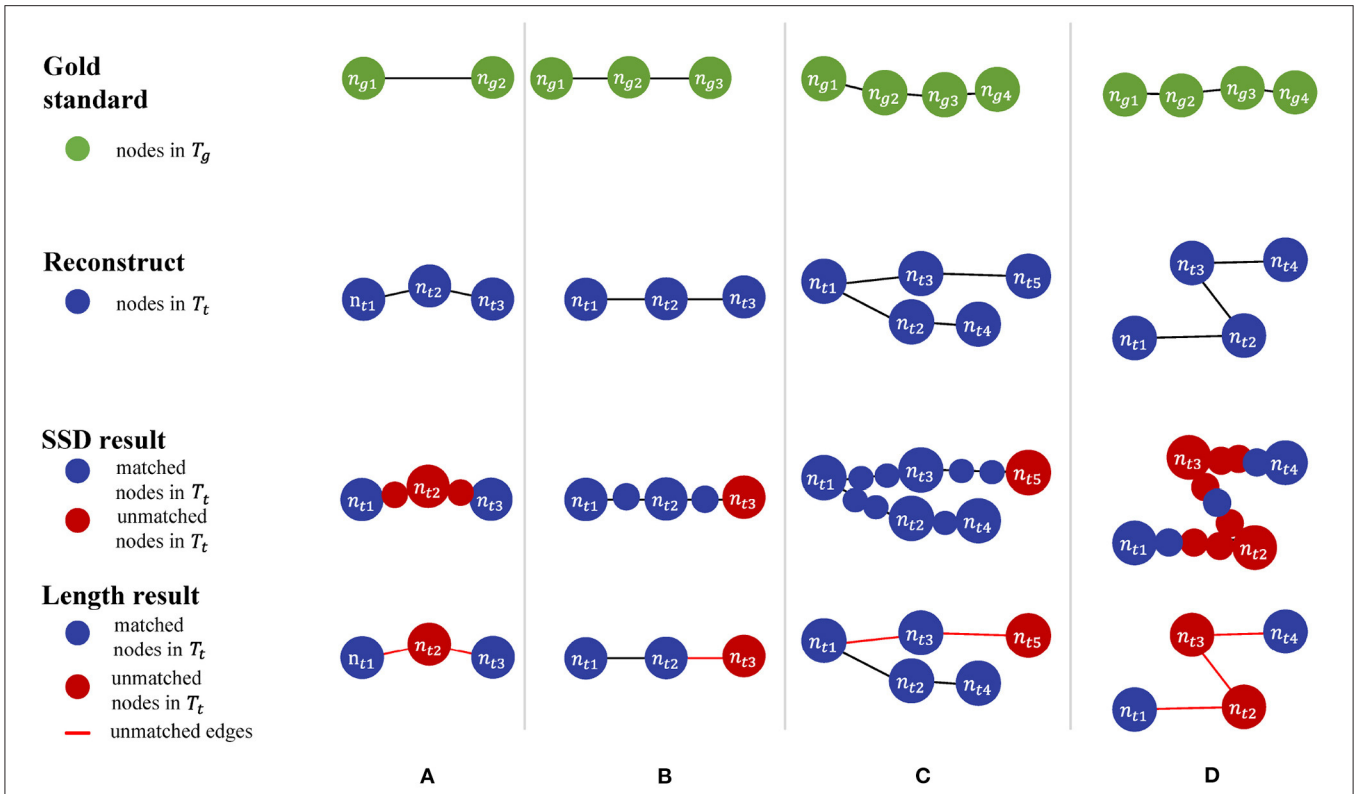

FIGURE 3 | (A–D) Four manually constructed cases for testing geometrical metrics.

TABLE 3 | PyNeval results of the SSD and length metrics for the geometrical cases are shown in **Figures 3A–D**.

Method	Index	File name			
		A	B	C	D
SSD metric	SSD score	1.66	1.60	0.51	1.49
	Recall	0.33	0.86	1.00	0.27
	Precision	0.36	0.83	0.95	0.18
	F1 score	0.35	0.85	0.98	0.21
Length metric	Recall	0.00	0.47	1.00	0.00
	Precision	0.00	0.50	0.54	0.00
	F1 score	0.00	0.48	0.70	0.00

the same tracing algorithm. We can treat this as a numerical optimization problem. For any tunable reconstruction program $\mathfrak{P}(I|\boldsymbol{\theta})$, in which image I and parameteres $\boldsymbol{\theta}$ are inputs and SWC model is the output, we define the optimization problem as

$$\min_{textbf\theta} \mathrm{E}(\mathcal{L}(\mathfrak{P}(I|\theta), T_g(I))|I) \tag{20}$$

where $\mathcal{L}$ is the loss function that can be computed from reconstruction metrics.

In real applications, we expect parameters optimized on a training dataset can be generalized to other images from the same imaging protocol. Therefore, we carried out a cross-validation experiment on four image blocks (**Figure 5**) from a whole mouse brain sample acquired by fMOST (Gong et al., 2016). The cross-validation searched for the best parameters for each block and used these optimized parameters to trace other blocks. In our experiment, we used the F1 score of the SSD metric as the loss function to optimize the automatic neuron tracing method used in neuTube (Zhao et al., 2011), which has two numerical parameters for adjusting the sensitivity of branch detection. The optimization process was performed by simulated annealing (Van Laarhoven and Aarts, 1987), which searches the parameters iteratively. A new parameter $\theta^{(k+1)}$ at the kth iteration was calculated by

$$\theta^{(k+1)} = \theta^{(k)} + 20\frac{u}{|u|} * t_k * ((1 + \frac{1}{t_k})^{|u|} - 1) \tag{21}$$

where u was drawn randomly from $[-1, 1]\backslash 0$ and t_k was the temperature at the kth iteration. Starting from $t_1 = 0.01$, the temperature was decreased every 25 iterations at the rate of 0.96. The stop criterion was that the temperature was below 10^{-5} or the optimal value had not been improved for 20 iterations.

The results show that the optimized parameters outperformed the default parameters consistently, no matter which image block was used in parameter searching (**Figure 6**), presenting a successful example of using PyNeval in improving automatic neuron tracing.

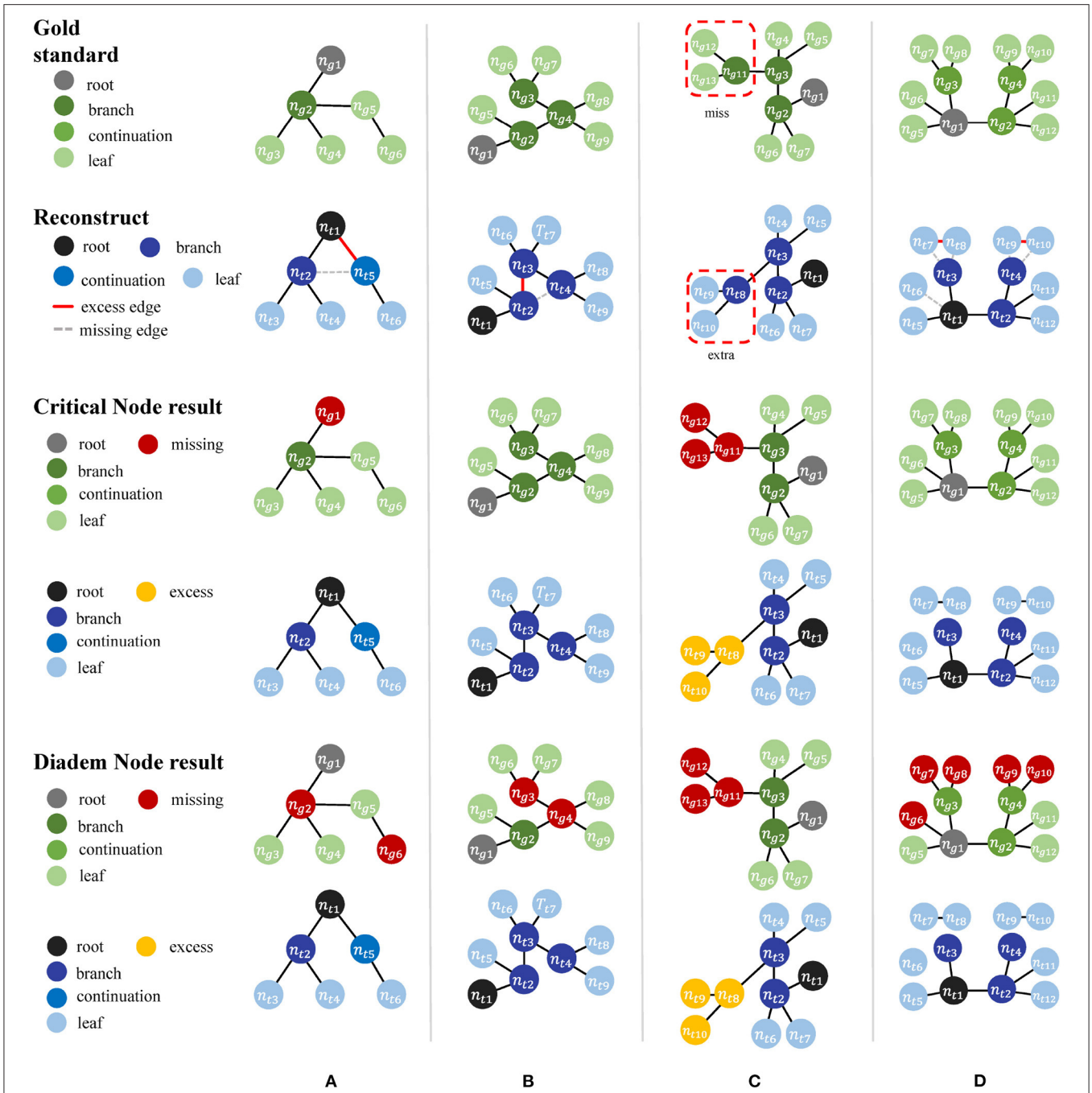

FIGURE 4 | (A–D) Four manually constructed cases for testing topological metrics.

TABLE 4 | PyNeval results of the DIADEM and length metrics for the topological cases are shown in **Figures 4A–D**.

Method	Index	file name			
		A	B	C	D
Diadem metric	Score	0.625	0.56	0.69	0.72
Critical node metric	Recall	0.80	1.00	0.70	1.00
	Precision	1.00	1.00	0.70	1.00
	F1 score	0.89	1.00	0.70	1.00

4. CONCLUSION AND FUTURE WORK

Motivated by the difficulties of evaluating automatic neuron tracing methods, we have developed PyNeval, a user-friendly Python toolbox to help method developers focus on algorithm development and method users choose a proper method for their own applications. PyNeval has made four popular metrics that cover both the geometrical and topological categories easily accessible to the community. A user can easily install PyNeval through common Python package managers and run the

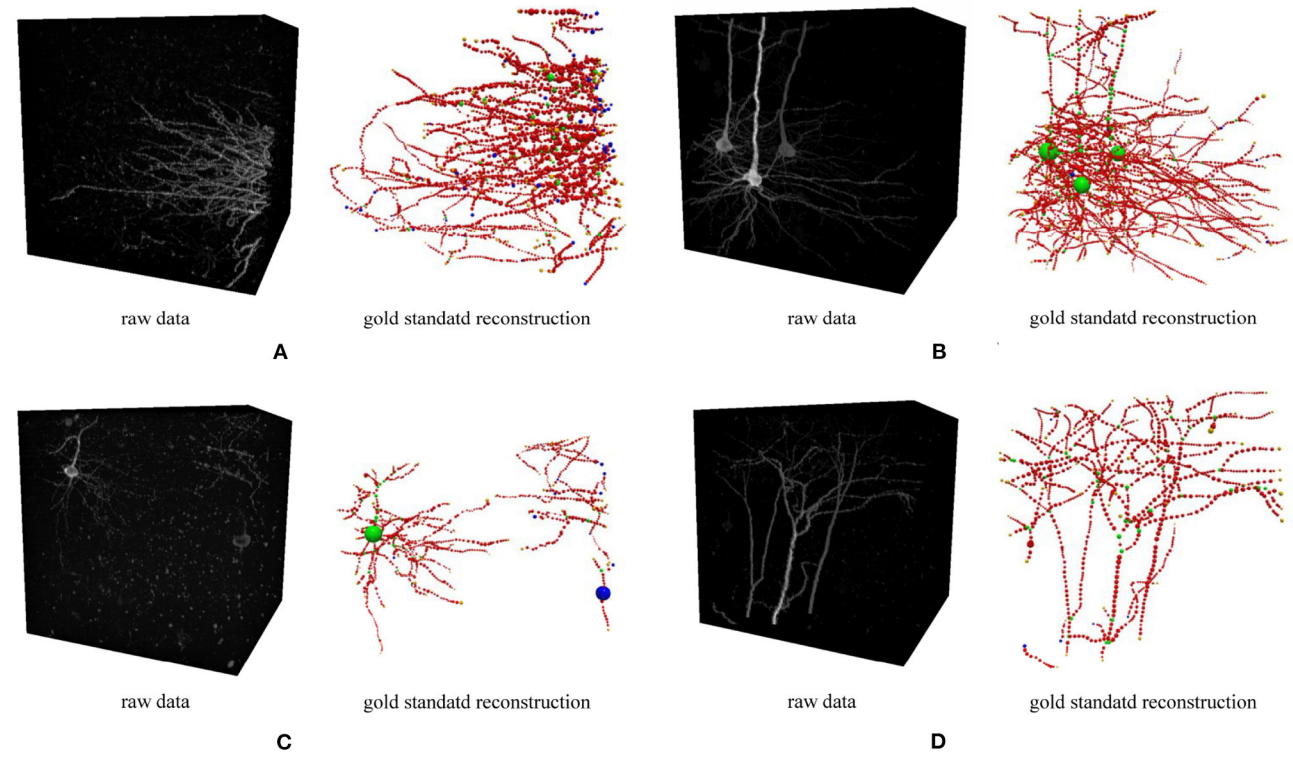

FIGURE 5 | Four 3D neuron images used in the optimization experiment. Each image and its gold standard reconstruction is rendered side by side in each panel labeled by the corresponding dataset ID. **(A)** FM3, **(B)** FM4, **(C)** FM5, and **(D)** FM6.

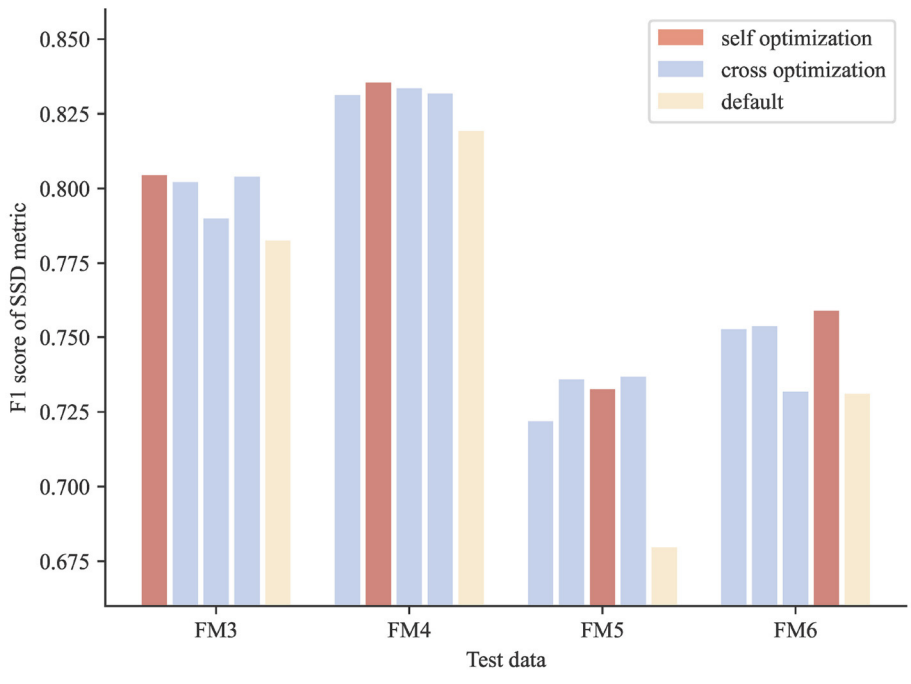

FIGURE 6 | Cross-validation results of parameter optimization for neuron reconstruction. The scores of the optimized parameters are consistently better than those of the default parameters for all the test images.

program as a command line with a straightforward but flexible interface. We have also shared the source code of PyNeval on https://github.com/CSDLLab/PyNeval to show how the metrics were implemented exactly as well as inspire further development.

To facilitate further development, PyNeval has a well-modularized architecture for maximizing its extensibility. It is straightforward to add more metrics such as the NetMets metric (Mayerich et al., 2012) in the future while keeping backward compatibility. Another important plan for further development is to make PyNeval an easy-to-use Python library as well, so that, other users can easily call functions in PyNeval from Python code directly, or even contribute their own metrics to PyNeval.

DATA AVAILABILITY STATEMENT

The raw data supporting the conclusions of this article will be made available by the authors, without undue reservation.

ETHICS STATEMENT

The animal study was reviewed and approved by Zhejiang University.

AUTHOR CONTRIBUTIONS

TZ and NZ designed and supervised the project. HZ wrote most part of the software with help from YY and TZ. HZ, CL, and JD performed data analysis. HZ, TZ, and NZ wrote the manuscript. All authors contributed to the article and approved the submitted version.

FUNDING

This work is supported by the National Key R&D Program of China (2020YFB1313501), Zhejiang Provincial Natural Science Foundation (LR19F020005), National Natural Science Foundation of China (61972347, 61976089), and Hunan Provincial Science & Technology Project Foundation (2018RS3065, 2018TP1018).

ACKNOWLEDGMENTS

We thank Wenzhi Sun and Wei Wu for providing fMOST data.

REFERENCES

Acciai, L., Soda, P., and Iannello, G. (2016). Automated neuron tracing methods: an updated account. *Neuroinformatics* 14, 353–367. doi: 10.1007/s12021-016-9310-0

Anytree. (2020). https://pypi.org/project/anytree/ (accessed Auguest 31, 2021).

Bille, P. (2005). A survey on tree edit distance and related problems. *Theor. Comput. Sci.* 337, 217–239. doi: 10.1016/j.tcs.2004.12.030

Cannon, R. C., Turner, D. A., Pyapali, G. K., and Wheal, H. V. (1998). An on-line archive of reconstructed hippocampal neurons. *J. Neurosci Methods* 84, 49–54. doi: 10.1016/S0165-0270(98)00091-0

Feng, L., Zhao, T., and Kim, J. (2015). neutube 1.0: a new design for efficient neuron reconstruction software based on the swc format. *eNeuro* 2:ENEURO.0049-14.2014. doi: 10.1523/ENEURO.0049-14.2014

Gillette, T. A., Brown, K. M., and Ascoli, G. A. (2011a). The diadem metric: comparing multiple reconstructions of the same neuron. *Neuroinformatics* 9, 233–245. doi: 10.1007/s12021-011-9117-y

Gillette, T. A., Brown, K. M., Svoboda, K., Liu, Y., and Ascoli, G. A. (2011b). Diademchallenge. org: a compendium of resources fostering the continuous development of automated neuronal reconstruction. *Neuroinformatics* 9, 303–304. doi: 10.1007/s12021-011-9104-3

Gong, H., Xu, D., Yuan, J., Li, X., Guo, C., Peng, J., et al. (2016). High-throughput dual-colour precision imaging for brain-wide connectome with cytoarchitectonic landmarks at the cellular level. *Nat. Commun.* 7:12142. doi: 10.1038/ncomms12142

Halavi, M., Hamilton, K. A., Parekh, R., and Ascoli, G. (2012). Digital reconstructions of neuronal morphology: three decades of research trends. *Front. Neurosci.* 6:49. doi: 10.3389/fnins.2012.00049

KDtree. (2017). https://pypi.org/project/kdtree/ (accessed Auguest 31, 2021).

Mayerich, D., Bjornsson, C., Taylor, J., and Roysam, B. (2012). Netmets: software for quantifying and visualizing errors in biological network segmentation. *BMC Bioinformatics* 13, S7. doi: 10.1186/1471-2105-13-S8-S7

Oliphant, T. E. (2007). Python for scientific computing. *Comput. Sci. Eng.* 9, 10–20. doi: 10.1109/MCSE.2007.58

Parekh, R., and Ascoli, G. A. (2013). Neuronal morphology goes digital: a research hub for cellular and system neuroscience. *Neuron* 77, 1017–1038. doi: 10.1016/j.neuron.2013.03.008

Peng, H., Hawrylycz, M., Roskams, J., Hill, S., Spruston, N., Meijering, E., et al. (2015). Bigneuron: large-scale 3d neuron reconstruction from optical microscopy images. *Neuron* 87, 252–256. doi: 10.1016/j.neuron.2015.06.036

Peng, H., Long, F., Zhao, T., and Myers, E. (2011). Proof-editing is the bottleneck of 3d neuron reconstruction: the problem and solutions. *Neuroinformatics* 9, 103–105. doi: 10.1007/s12021-010-9090-x

Peng, H., Ruan, Z., Long, F., Simpson, J. H., and Myers, E. W. (2010). V3d enables real-time 3d visualization and quantitative analysis of large-scale biological image data sets. *Nat. Biotechnol.* 28, 348–353. doi: 10.1038/nbt.1612

Rtree. (2020). https://pypi.org/project/rtree/ (accessed Auguest 31, 2021).

Van Der Walt, S., Colbert, S. C., and Varoquaux, G. (2011). The numpy array: a structure for efficient numerical computation. *Comput. Sci. Eng.* 13, 22–30. doi: 10.1109/MCSE.2011.37

Van Laarhoven, P. J., and Aarts, E. H. (1987). "Simulated annealing," in *Simulated annealing: Theory and applications* (Berlin: Springer), 7–15.

Wang, Y., Narayanaswamy, A., Tsai, C.-L., and Roysam, B. (2011). A broadly applicable 3-d neuron tracing method based on open-curve snake. *Neuroinformatics* 9, 193–217. doi: 10.1007/s12021-011-9110-5

Zhao, T., Xie, J., Amat, F., Clack, N., Ahammad, P., Peng, H., et al. (2011). Automated reconstruction of neuronal morphology based on local geometrical and global structural models. *Neuroinformatics* 9, 247–261. doi: 10.1007/s12021-011-9120-3

Conflict of Interest: The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Publisher's Note: All claims expressed in this article are solely those of the authors and do not necessarily represent those of their affiliated organizations, or those of the publisher, the editors and the reviewers. Any product that may be evaluated in this article, or claim that may be made by its manufacturer, is not guaranteed or endorsed by the publisher.