Rebuttal for ‘Guided Proofreading of Automatic Segmentations for Connectomics’

Thank you for your time and constructive comments. We will fix all minor issues; we address the major issues below.

R2: Quantitative Evaluation R2 requests an objective quantitative evaluation. Our ‘automatic’ and ‘oracle’ experiments are exactly this. We define such experiments in lines 573–590 and report the results in Fig. 6, Fig. 7, and lines 792–818 (also in supplemental Sec. 2 and 3). These evaluations use the quantitative VI metric against a ground truth segmentation, with no user in the loop.

R2: Faster than State of the Art? ‘Faster’ here considers both how long each correction takes, and the likely VI reduction from that correction. Our approach has comparable correction time to the state-of-the-art Focused Proofreading approach, but a significantly greater VI reduction per correction (7.5×). We agree that the presentation of this comparison was not ideal. We will add Table 1 to make this clearer (previously reported in lines 756-765, slopes in Fig. 6, and column 3 in Fig. 7).

Table 1: Average proofreading speed for users of Dojo, Focused Proofreading (FP) and our Guided Proofreading (GP). For comparable correction time, our system achieves significantly higher VI reduction per minute (7.5×) over state-of-the-art FP.

Approach	Time Per Correction (s)	VI Reduction Per Minute
Dojo	30.5	-0.002
FP	4.9	0.00023
GP	6.2	0.00173

R2: Reproducibility R2 expresses concerns regarding reproducibility. As per line 847, we have promised to release all our code including trained models and data, and we disclose all parameters.

R2: How were Optimal Parameters chosen? The **threshold** $p_t = 0.95$ was observed to be stable when evaluating VI reduction on previously-unseen test data (lines 585-586, supplemental Sec. 1.3). The **input border is dilated by 5 pixels** to consider slight edge ambiguities and extra-cellular space in high-resolution electron microscopy data (lines 308-310). During merge error detection, **labels are dilated by 20 pixels** prior to finding potential borders (line 323) with border-seeded watershed—this way the borders tend to attach to real membrane boundaries (lines 364-366). As we use a CNN, there are many additional parameters; we consider choosing these effectively to be an open problem, and for this we used learned experience and local brute-force searches on separate training data.

R3: Training Datasets—U-net vs. GP? R3 raises the question of whether our GP approach was trained on the same data as membrane detection (U-net). There was no overlap (Tab. 2).

Table 2: Training data of membrane detection vs. training data of GP vs. test data (for supplemental material).

Training Set	Training Set	Test Set
U-Net / GALA	GP	GP
AC3+AC4	L. Cylinder	L. Cylinder$_{\text{test}}$
(1024 × 1024 × 175vx)	(2048 × 2048 × 250vx)	(2048 × 2048 × 50vx)
AC4 excl. test	L. Cylinder	AC4$_{\text{test}}$ subvolume
(1024 × 1024 × 90vx)	(2048 × 2048 × 250vx)	(400 × 400 × 10vx)
AC3+AC4	CREMI A/B/C	CREMI A/B/C$_{\text{test}}$
(1024 × 1024 × 175vx)	(1250 × 1250 × 300vx)	(1250 × 1250 × 15vx)

R3: Merge Error Detection R3 requests a better explanation of the merge error detection (Sec. 3.2). We have updated Fig. 4 in the main paper to be more clear (Fig. 1). We will also add pseudo code of the algorithm to the supplemental material to promote understanding (Alg. 1).

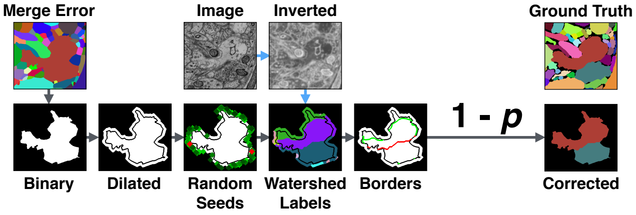

Figure 1: Potential borders are generated on inverted images by randomly placing watershed seeds (green) on the boundary of a dilated segment. The best ranked seeds and border (both in red) result in the shown error correction.

Algorithm 1 Merge Error Detection for a label l

```
$l_d$ = dilate($l$, 20)
invImage = invert(image)
for N iterations do
    $s_1$, $s_2$ = randomSeedsOnBoundary($l_d$)
    wsImage = watershed(invImage, $l_d$, $s_1$, $s_2$)
    border = border(wsImage)
    $p$ = rank(border)
    $p_{\text{merge}} = 1 - p$
find(max $p_{\text{merge}}$)
```

R3: GALA Active Learning Classifier In our automatic segmentation pipeline (line 499), GALA uses a random forest classifier to agglomerate segments. We used an agglomeration level of 0.3 (after a grid search). We will add a section to the supplemental material containing a full description of the approach and our use of it.