# Cell2Doc Supplementary Material

August 18, 2023

## 1 Cell2Doc Outputs on Sample Examples

Figures 1 , 2, 3 and 4 show output documentation on sample code examples for PLBART [1], CodeBERT [2], UnixCoder [3], and GraphCodeBERT [4] respectively. We can see from these examples that with Cell2Doc pipeline, all these pre-trained models have provided longer and more informative documentation compared to their base stand-alone versions (shown in the previous columns in the table). These observations highlight that our Cell2Doc pipeline is model agnostic and can improve the effectiveness of other pre-trained models.

## 2 Cell2Doc Human Evaluation with CodeBERT

We have done the human evaluation of Cell2Doc using CodeT5 and CodeBERT as the CoDoc model. As CodeBERT is an encoder-only model, we have used a 6-layer transformer decoder with it to build the CoDoc model. We have only used CodeT5 in the main paper because it has performed better than CodeBERT overall. Figure 5 summarizes the human evaluation results using CodeBERT in CoDoc of Cell2Doc. Like CodeT5 [5], we can see that Cell2Doc also improves CodeBERT's effectiveness across all three dimensions.

## 3 Other Input Representations

Inline code comments can provide additional information about the code, which can be utilized to create better documentation. We tested this input representation idea with the CodeBERT model in CoDoc, which can take both code and natural language as input. Code and comments present in a code cell can be represented as $[code\_snippet_1, comment_1, code\_snippet_2, comment_2, \cdots code\_snippet_n]$ where $code\_snippet_i$ are code segments (present between two consecutive comments) and $comment_i$ are the comments. We use it as input to CodeBERT by using a delimiter between code and comment. As target documentation, we use only summarized markdown in these cases. The BLEU [6] and ROUGE [7] (only F1 for ROUGE-1, ROUGE-2 and ROUGE-L) scores for this experiment are **20.37** and (**25.29**, **11.60**, and **28.24**) respectively. Compared with the automated evaluation where CodeBERT is used in CoDoc, we can see that the numbers are better when comments are included in the input with respect to other baselines. It does not beat the SCSCM representation, though, and the reason for it might be that SCSCM has more data points in the training set as it uses comments as separate labels, and comments serve as good quality labels as well.

| Code cell | PLBART | Cell2Doc (PLBART) |
| --- | --- | --- |
| `n_classes = 12`<br>`data, labels = make_classification(n_samples=2000, n_features=100, n_informative=50, n_classes=n_classes, random_state=random_state)`<br>`X, X_test, y, y_test = train_test_split(data, labels, test_size=0.2, random_state=random_state)`<br>`X_train, X_valid, y_train, y_valid = train_test_split(X, y, test_size=0.25, random_state=random_state)`<br>`print('Data shape:')`<br>`print('X_train: %s, X_valid: %s, X_test: %s\n' %(X_train.shape, X_valid.shape, X_test.shape))` | Splitting the data into training and validation data | STEP 1: Number of classes<br>STEP 2: Create classification object<br>STEP 3: taking holdout set for validating with random state<br>STEP 4: Spliting the dataset into training and validation<br>STEP 5: Check data shape |
| `def cel(y_true, y_pred):`<br>`    y_true = torch.argmax(y_true, axis=-1)`<br>`    return nn.CrossEntropyLoss()(y_pred, y_true.squeeze())`<br>`def accuracy(y_true, y_pred):`<br>`    y_true = torch.argmax(y_true, axis=-1).squeeze()`<br>`    y_pred = torch.argmax(y_pred, axis=-1).squeeze()`<br>`    return (y_true == y_pred).float().sum()/len(y_true)` | Evaluation of the Model | STEP 1: The function for evaluation is borrowed from<br>STEP 2: Cross entropy loss |
| `train_images = []`<br>`image_dirs = np.take(os.listdir('../input/train'), select_rows)`<br>`for image_dir in tqdm(sorted(image_dirs)):`<br>`    image = imread('../input/train/'+image_dir)`<br>`    train_images.append(image)`<br>`    del image`<br>`    gc.collect()`<br>`train_images = np.array(train_images)` | Read in the train images | STEP 1: Read in the training images<br>STEP 2: Read in all the images<br>STEP 2.1: Read in image and save in list<br>STEP 2.2: Delete to reduce memory usage |

Figure 1: Code and generated documentation examples using PLBART in CoDoc of Cell2Doc

| Code cell | CodeBERT | Cell2Doc (CodeBERT) |
| --- | --- | --- |
| `n_classes = 12`<br>`data, labels = make_classification(n_samples=2000, n_features=100, n_informative=50, n_classes=n_classes, random_state=random_state)`<br>`X, X_test, y, y_test = train_test_split(data, labels, test_size=0.2, random_state=random_state)`<br>`X_train, X_valid, y_train, y_valid = train_test_split(X, y, test_size=0.25, random_state=random_state)`<br>`print('Data shape:')`<br>`print('X_train: %s, X_valid: %s, X_test: %s\n' %(X_train.shape, X_valid.shape, X_test.shape))` | Train a Classifier | STEP 1: Number of classes per image<br>STEP 2: Create a Classification object<br>STEP 3: Train Validation Split<br>STEP 4: Split the dataset into training and validation<br>STEP 5: Checking the shape of the data |
| `def cel(y_true, y_pred):`<br>`    y_true = torch.argmax(y_true, axis=-1)`<br>`    return nn.CrossEntropyLoss()(y_pred, y_true.squeeze())`<br>`def accuracy(y_true, y_pred):`<br>`    y_true = torch.argmax(y_true, axis=-1).squeeze()`<br>`    y_pred = torch.argmax(y_pred, axis=-1).squeeze()`<br>`    return (y_true == y_pred).float().sum()/len(y_true)` | Cross entropy loss | STEP 1: Cross entropy loss<br>STEP 2: The accuracy of the classifier |
| `train_images = []`<br>`image_dirs = np.take(os.listdir('../input/train'), select_rows)`<br>`for image_dir in tqdm(sorted(image_dirs)):`<br>`    image = imread('../input/train/'+image_dir)`<br>`    train_images.append(image)`<br>`    del image`<br>`    gc.collect()`<br>`train_images = np.array(train_images)` | load train images | STEP 1: Get train images<br>STEP 2: Read in each image<br>STEP 2.1: Read in the image<br>STEP 2.2: Delete image to reduce memory usage |

Figure 2: Code and generated documentation examples using CodeBERT in CoDoc of Cell2Doc

| Code cell | UnixCoder | Cell2Doc (UnixCoder) |
|---|---|---|
| n_classes = 12<br>data, labels = make_classification(n_samples=2000, n_features=100, n_informative=50, n_classes=n_classes, random_state=random_state)<br>X, X_test, y, y_test = train_test_split(data, labels, test_size=0.2, random_state=random_state)<br>X_train, X_valid, y_train, y_valid = train_test_split(X, y, test_size=0.25, random_state=random_state)<br>print('Data shape:')<br>print('X_train: %s, X_valid: %s, X_test: %s \n' %(X_train.shape, X_valid.shape, X_test.shape)) | Prepare data for modeling | STEP 1: Number of classes used for visualization<br>STEP 2: Create data and labels for model<br>STEP 3: Split the data into train and test data<br>STEP 4: Split the dataset into training and validation sets<br>STEP 5: Check data types |
| def cel(y_true, y_pred):<br>    y_true = torch.argmax(y_true, axis=-1)<br>    return nn.CrossEntropyLoss()(y_pred, y_true.squeeze())<br>def accuracy(y_true, y_pred):<br>    y_true = torch.argmax(y_true, axis=-1).squeeze()<br>    y_pred = torch.argmax(y_pred, axis=-1).squeeze()<br>    return (y_true == y_pred).float().sum()/len(y_true) | Cross entropy loss | STEP 1: Cross entropy loss<br>STEP 2: Defining the accuracy |
| train_images = []<br>image_dirs = np.take(os.listdir('../input/train'), select_rows)<br>for image_dir in tqdm(sorted(image_dirs)):<br>    image = imread('../input/train/'+image_dir)<br>    train_images.append(image)<br>    del image<br>    gc.collect()<br>train_images = np.array(train_images) | Read and shuffle filenames | STEP 1: Read in the training dataset<br>STEP 2: Read in images<br>STEP 2.1: Read in the training images<br>STEP 2.2: Free up some memory |

Figure 3: Code and generated documentation examples UnixCoder in CoDoc of Cell2Doc

| Code cell | GraphCodeBERT | Cell2Doc (GraphCodeBERT) |
|---|---|---|
| n_classes = 12<br>data, labels = make_classification(n_samples=2000, n_features=100, n_informative=50, n_classes=n_classes, random_state=random_state)<br>X, X_test, y, y_test = train_test_split(data, labels, test_size=0.2, random_state=random_state)<br>X_train, X_valid, y_train, y_valid = train_test_split(X, y, test_size=0.25, random_state=random_state)<br>print('Data shape:')<br>print('X_train: %s, X_valid: %s, X_test: %s \n' %(X_train.shape, X_valid.shape, X_test.shape)) | Splitting the dataset into train and validation set | STEP 1: Define some constants<br>STEP 2: Use data augmentation<br>STEP 3: Splitting the into train and test data<br>STEP 4: Split datas in train and validation set<br>STEP 5: Overview of the data |
| def cel(y_true, y_pred):<br>    y_true = torch.argmax(y_true, axis=-1)<br>    return nn.CrossEntropyLoss()(y_pred, y_true.squeeze())<br>def accuracy(y_true, y_pred):<br>    y_true = torch.argmax(y_true, axis=-1).squeeze()<br>    y_pred = torch.argmax(y_pred, axis=-1).squeeze()<br>    return (y_true == y_pred).float().sum()/len(y_true) | NumtaDB Classification Report | STEP 1: Define loss function<br>STEP 2: define eval metrics |
| train_images = []<br>image_dirs = np.take(os.listdir('../input/train'), select_rows)<br>for image_dir in tqdm(sorted(image_dirs)):<br>    image = imread('../input/train/'+image_dir)<br>    train_images.append(image)<br>    del image<br>    gc.collect()<br>train_images = np.array(train_images) | Load the data | STEP 1: List of images<br>STEP 2: Iterate over all images<br>STEP 2.1: Read in the training data<br>STEP 2.2: Delete unnecessary images |

Figure 4: Code and generated documentation examples using GraphCodeBERT in CoDoc of Cell2Doc

| Index | Model | Correctness | Informativeness | Readability |
|---|---|---|---|---|
| 1 | CodeBERT (CM) | $\mu = 3.00, \sigma = 1.34$ | $\mu = 2.8, \sigma = 1.24$ | $\mu = 3.65, \sigma = 1.18$ |
| 2 | CodeBERT (CSM) | $\mu = 2.74, \sigma = 1.39$ | $\mu = 2.73, \sigma = 1.29$ | $\mu = 3.62, \sigma = 1.14$ |
| 3 | CodeBERT (ECSM) | $\mu = 2.57, \sigma = 1.31$ | $\mu = 2.57, \sigma = 1.28$ | $\mu = 3.62, \sigma = 1.11$ |
| 4 | CodeBERT (SCSCM) | $\mu = 3.10, \sigma = 1.30$ | $\mu = 3.04, \sigma = 1.19$ | $\mu = 3.72, \sigma = 1.10$ |
| 5 | CodeBERT (Cell2Doc) | $\mu = \mathbf{3.81}, \sigma = 1.13$ | $\mu = \mathbf{4.00}, \sigma = 1.11$ | $\mu = \mathbf{4.22}, \sigma = 0.90$ |

Figure 5: Results of the human evaluation using CodeBERT in CoDoc of Cell2Doc

# References

[1] W. Ahmad, S. Chakraborty, B. Ray, and K.-W. Chang, "Unified pre-training for program understanding and generation," in *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Online: Association for Computational Linguistics, Jun. 2021, pp. 2655–2668. [Online]. Available: https://aclanthology.org/2021.naacl-main.211

[2] Z. Feng, D. Guo, D. Tang, N. Duan, X. Feng, M. Gong, L. Shou, B. Qin, T. Liu, D. Jiang, and M. Zhou, "CodeBERT: A pre-trained model for programming and natural languages," in *Findings of the Association for Computational Linguistics: EMNLP 2020*. Online: Association for Computational Linguistics, Nov. 2020, pp. 1536–1547. [Online]. Available: https://aclanthology.org/2020.findings-emnlp.139

[3] D. Guo, S. Lu, N. Duan, Y. Wang, M. Zhou, and J. Yin, "Unixcoder: Unified cross-modal pre-training for code representation," 2022. [Online]. Available: https://arxiv.org/abs/2203.03850

[4] D. Guo, S. Ren, S. Lu, Z. Feng, D. Tang, S. Liu, L. Zhou, N. Duan, A. Svyatkovskiy, S. Fu *et al.*, "Graphcodebert: Pre-training code representations with data flow," *arXiv preprint arXiv:2009.08366*, 2020.

[5] Y. Wang, W. Wang, S. Joty, and S. C. H. Hoi, "Codet5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation," 2021. [Online]. Available: https://arxiv.org/abs/2109.00859

[6] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, "Bleu: a method for automatic evaluation of machine translation," in *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*. Philadelphia, Pennsylvania, USA: Association for Computational Linguistics, Jul. 2002, pp. 311–318. [Online]. Available: https://aclanthology.org/P02-1040

[7] C.-Y. Lin, "ROUGE: A package for automatic evaluation of summaries," in *Text Summarization Branches Out*. Barcelona, Spain: Association for Computational Linguistics, Jul. 2004, pp. 74–81. [Online]. Available: https://aclanthology.org/W04-1013