1 Evaluation Tasks

We conduct experiments on three representative code intelligence tasks: code summarization, defect detection, and assertion generation, for covering different task types, i.e., Code → Text, Code → Label, and Code → Code.

1.1 Code Summarization

Code Summarization aims to generate useful comments for a given code snippet. It can help alleviate the developers' cognitive efforts in comprehending programs [Garousi et al.(2015), Chen and Huang(2009)].

Datasets. In this study, we conduct experiments on two popular benchmark datasets JCSD and PCSD, which contain Java and Python source code, respectively. The JCSD dataset we used is publicly released by Hu et al. [Hu et al.(2018)], which contains 87,136 pairs of Java methods and comments collected from 9,714 GitHub repositories. The PCSD dataset comprises 92,545 functions with their respective documentation, which is originally collected by Barone et al. [Barone and Sennrich(2017)] and later processed by Wei et al. [Wei et al.(2020)]. For our experiments, we directly used the benchmark datasets released by previous studies [Hu et al.(2018), Ahmad et al.(2020)], in which the datasets are divided into training, validation, and test sets in a ratio of 8 : 1 : 1 and 6 : 2 : 2 for Java and Python, respectively. As reported in previous work [Shi et al.(2022b), Mu et al.(2022)], there are duplicated data in the training and test set of the JCSD dataset. Therefore, following them, we remove the test samples that also appear in the training or validation set and finally get a deduplicated test set with 6,489 samples. Since there has been no dataset for the evaluation of code intelligence tasks in a semi-supervised setting, we propose to simulate it by extending existing datasets. Specifically, following previous studies [Mi et al.(2021), Ke et al.(2022)], we randomly dividing the initial training data into two subsets: labeled training data and an unlabeled dataset, with the ratio of 9:1.

Metrics. For code summarization, we follow previous work [Ahmad et al.(2020), Mu et al.(2022), Shi et al.(2022a)] and use four popular metrics BLEU-4 [Papineni et al.(2002)], ROUGE-L [Lin(2004)], METEOR [Banerjee and Lavie(2005)], and CIDEr [Vedantam et al.(2015)] for evaluation.

BLEU measures the similarity of two summaries by calculating the ratio of N groups of word similarity between them. A higher BLEU score indicates higher similarity. We follow previous work [Ahmad et al.(2020), Zhang et al.(2020)] and use BLEU-4 for evaluation. It is computed as:

$$BLEU-4 = BP \times \exp(\sum_{n=1}^{4} \tau_n \log P_n), \tag{1}$$

where P_n is the ratio of n-gram in the prediction summary that are also in the reference summary. BP is the brevity penalty and τ_n is set to 1/4.

METEOR evaluates generated summaries by aligning them to the reference summaries and calculating the similarity scores as follows:

$$METEOR = (1 - \gamma \cdot \mathit{frag}^{\beta}) \cdot \frac{P \cdot R}{\alpha \cdot P + (1-\alpha) \cdot R}, \tag{2}$$

where P and R are unigram precision and recall, *frag* is the fragmentation fraction. α, β and γ are three penalty parameters whose default values are 0.9, 3.0, and 0.5, respectively.

ROUGE-L calculates the F-score based on Longest Common Subsequence (LCS) between two summaries. Given a generated summary X and the reference summary Y, ROUGE-L is computed as:

$$P_{lcs} = \frac{LCS(X,Y)}{n}, \quad R_{lcs} = \frac{LCS(X,Y)}{m}, \tag{3}$$

$$F_{lcs} = \frac{(1+\beta^2) P_{lcs} R_{lcs}}{R_{lcs} + \beta^2 P_{lcs}}, \tag{4}$$

where m and n are the length of X and Y, respectively. $\beta = P_{lcs}/R_{lcs}$ and F_{lcs} is the computed ROUGE-L score.

CIDEr considers the frequency of n-grams in the reference sentences by computing the TF-IDF weighting for each n-gram. CIDEr$_n$score for n-gram is computed using the average cosine similarity between the candidate sentence and the reference.

```
# Code from the test set:
def print_bucket_acl_for_user(bucket_name, user_email):
    storage_client = storage.Client()
    bucket = storage_client.bucket(bucket_name)
    bucket.acl.reload()
    roles = bucket.acl.user(user_email).get_roles()
    print roles
```

```
Summary generated by Unixcoder:
print the current users name for the specified user.
Summary generated by Unixcoder+HINT:
prints the name for the specified bucket and user.
Ground truth summary:
prints out a buckets access control list for a given user.
```

Figure 1: Error case on the code summarization task.

1.2 Defect Detection

Defect detection aims at identifying the vulnerabilities in the given program, which is crucial to defend a software system from cyberattack [Fan et al.(2020), Zhou et al.(2019)].

Datasets. In our experiments, we utilize the widely-used Big-Vul dataset created by Fan et al. [Fan et al.(2020)]. This dataset contains C/C++ code snippets sourced from more than 300 GitHub projects dating from 2002 to 2019 in Common Vulnerabilities and Exposures (CVE) database. Following previous studies [Fan et al.(2020)], we partition the dataset into training, validation, and test sets with a ratio of 8:1:1. Same with code summarization, we also further construct the labeled training data and unlabeled data by dividing the original training set of Big-Vul with a ratio of 1:9.

Metrics. For vulnerability detection, we follow previous work [Zhou et al.(2019), Li et al.(2021)] and evaluate the results by Precision (P), Recall (R), and F1.

$$P = \frac{TP}{TP + FP}, R = \frac{TP}{TP + FN}, F1 = \frac{2 \cdot P \cdot R}{P + R} \tag{5}$$

$$ACC = \frac{TP + TN}{TP + TN + FN + FP} \tag{6}$$

where TP, FP, TN, and FN denote the number of true positives, false positives, true negatives, and false negatives respectively.

1.3 Assertion Generation

Assertion Generation is the task of automatically generating meaningful assert statements for unit tests. It can reduce the manual efforts in writing test cases and facilitate faster detection and diagnosis of software failures [Yu et al.(2022), Mastropaolo et al.(2021), Watson et al.(2020)].

Datasets. For assertion generation, we follow previous work [Yu et al.(2022), Mastropaolo et al.(2021)] and use the ATLAS dataset [Watson et al.(2020)]. It contains 188,154 real-world test assertions obtained from open-source projects in GitHub. The dataset is composed of eight categories of assertions, and each sample in ATLAS is comprised of a focal method and a test method which serve as the context for generating a single assertion for the given test method. We use the original partition of ATLAS and split it into three subsets: training, validation, and test, in an 8:1:1 ratio. The construction of an unlabeled dataset for assertion generation is also the same as the above two tasks. We randomly extract 90% of the training data for the construction of the unlabeled dataset and use the remaining data as the labeled dataset.

Metrics. For assertion generation, we follow previous work [Yu et al.(2022), Nashid et al.(2023)] in this field and use Exact Match (EM), Longest Common Subsequence (LCS), and Edit Distance (ED) as evaluation metrics. EM measures the percentage of samples that assert statements generated by the model that are identical to the reference. LCS is the ratio of the longest common subsequence between the predicted assertion and the ground truth assertion. ED determines the number of edit operations (including addition, deletion, and modification) required for the inferred output to match the expected output. Different from the above metrics, lower ED indicates higher similarity and better performance.

```
Focal-test from test set:
testIdAccessor(){
    java.lang.Long id = 3L; instance.setId(id);
    "<AssertPlaceHolder>";}
getId(){
    return id;
}
```

```
Assertion generated by Unixcoder+HINT:
org.junit.Assert.assertThat(id, instance.getId());
Ground truth assertion:
org.junit.Assert.assertEquals(id, instance.getId());
```

```
Focal-test from Unlable dataset:
testGetName(){
    java.lang.String id = "id"; togglePanelItem.setId(id);
    "<AssertPlaceHolder>";}
getId(){
    return id;
}
```

```
Generated pseudo label:
org.junit.Assert.assertEquals(id, togglePanelItem.getId());
```

Figure 2: Error case on the assertion generation task.

2 Limitation of HINT

To gain a deeper understanding of HINT's behavior and limitations, we further investigate cases where HINT fails to make accurate predictions and conclude two possible limitations of HINT.

The first limitation pertains to HINT's inability to introduce additional knowledge and rectify factual knowledge errors. From the example in the above Figure 1, UniXcoder misinterprets the term "*bucket_acl*" as the name of a bucket and fails to rectify this misunderstanding even after additional training on pseudo-labeled data. This shows that without external feedback HINT is hard to identify and rectify the problem on factual knowledge, which also aligns with recent findings on the limited self-correction ability of large language models [Huang et al.(2023)]. To potentially alleviate this limitation, integrating factual knowledge into pre-trained code models via the interaction with a knowledge base or search engine could be further studied.

The second limitation of HINT is the reliance on the capacity of the base model. HINT aims at autonomously synthesizing more labeled data for model training. However, when the base model lacks sufficient capacity, the benefits of additional training data are diminished. As depicted in the Figure 2, despite the presence of many training samples in the pseudo-labeled data illustrating the usage of "*assertEquals*", UniXcoder still fails to learn this and erroneously generates "*assertThat*" for the given function. We attribute this limitation to the inherent constraints of the model's capacity and believe that it could be mitigated by using more advanced pre-trained code models.

References

[Ahmad et al.(2020)] Wasi Uddin Ahmad, Saikat Chakraborty, Baishakhi Ray, and Kai-Wei Chang. 2020. A Transformer-based Approach for Source Code Summarization. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020*. Association for Computational Linguistics, 4998–5007.

[Banerjee and Lavie(2005)] Satanjeev Banerjee and Alon Lavie. 2005. METEOR: An Automatic Metric for MT Evaluation with Improved Correlation with Human Judgments. In *Proceedings of the Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization@ACL 2005, Ann Arbor, Michigan, USA, June 29, 2005*. Association for Computational Linguistics, 65–72.

[Barone and Sennrich(2017)] Antonio Valerio Miceli Barone and Rico Sennrich. 2017. A parallel corpus of Python functions and documentation strings for automated code documentation and code generation. *arXiv preprint arXiv:1707.02275* (2017).

[Chen and Huang(2009)] Jie-Cherng Chen and Sun-Jen Huang. 2009. An empirical analysis of the impact of software development problem factors on software maintainability. *J. Syst. Softw.* 82, 6 (2009), 981–992.

[Fan et al.(2020)] Jiahao Fan, Yi Li, Shaohua Wang, and Tien N. Nguyen. 2020. A C/C++ Code Vulnerability Dataset with Code Changes and CVE Summaries. In *MSR '20: 17th International Conference on Mining Software Repositories, Seoul, Republic of Korea, 29-30 June, 2020*. ACM, 508–512.

[Garousi et al.(2015)] Golara Garousi, Vahid Garousi-Yusifoglu, Günther Ruhe, Junji Zhi, Mahmood Moussavi, and Brian Smith. 2015. Usage and usefulness of technical software documentation: An industrial case study. *Inf. Softw. Technol.* 57 (2015), 664–682.

[Hu et al.(2018)] Xing Hu, Ge Li, Xin Xia, David Lo, Shuai Lu, and Zhi Jin. 2018. Summarizing Source Code with Transferred API Knowledge. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden*. ijcai.org, 2269–2275.

[Huang et al.(2023)] Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song, and Denny Zhou. 2023. Large Language Models Cannot Self-Correct Reasoning Yet. *CoRR* abs/2310.01798 (2023).

[Ke et al.(2022)] Pei Ke, Haozhe Ji, Zhenyu Yang, Yi Huang, Junlan Feng, Xiaoyan Zhu, and Minlie Huang. 2022. Curriculum-Based Self-Training Makes Better Few-Shot Learners for Data-to-Text Generation. In *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI 2022, Vienna, Austria, 23-29 July 2022*. ijcai.org, 4178–4184.

[Li et al.(2021)] Yi Li, Shaohua Wang, and Tien N. Nguyen. 2021. Vulnerability detection with fine-grained interpretations. In *ESEC/FSE '21: 29th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Athens, Greece, August 23-28, 2021*. ACM, 292–303.

[Lin(2004)] Chin-Yew Lin. 2004. ROUGE: A Package for Automatic Evaluation of Summaries. In *Text Summarization Branches Out*. Association for Computational Linguistics, Barcelona, Spain, 74–81.

[Mastropaolo et al.(2021)] Antonio Mastropaolo, Simone Scalabrino, Nathan Cooper, David Nader Palacio, Denys Poshyvanyk, Rocco Oliveto, and Gabriele Bavota. 2021. Studying the usage of text-to-text transfer transformer to support code-related tasks. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 336–347.

[Mi et al.(2021)] Fei Mi, Wanhao Zhou, Lingjing Kong, Fengyu Cai, Minlie Huang, and Boi Faltings. 2021. Self-training Improves Pre-training for Few-shot Learning in Task-oriented Dialog Systems. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 7-11 November, 2021*. Association for Computational Linguistics, 1887–1898.

[Mu et al.(2022)] Fangwen Mu, Xiao Chen, Lin Shi, Song Wang, and Qing Wang. 2022. Automatic Comment Generation via Multi-Pass Deliberation. In *37th IEEE/ACM International Conference on Automated Software Engineering, ASE 2022, Rochester, MI, USA, October 10-14, 2022*. ACM, 14:1–14:12.

[Nashid et al.(2023)] Noor Nashid, Mifta Sintaha, and Ali Mesbah. 2023. Retrieval-Based Prompt Selection for Code-Related Few-Shot Learning. (2023).

[Papineni et al.(2002)] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a Method for Automatic Evaluation of Machine Translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics, July 6-12, 2002, Philadelphia, PA, USA*. ACL, 311–318.

[Shi et al.(2022b)] Ensheng Shi, Yanlin Wang, Lun Du, Junjie Chen, Shi Han, Hongyu Zhang, Dongmei Zhang, and Hongbin Sun. 2022b. On the Evaluation of Neural Code Summarization. In *44th IEEE/ACM 44th International Conference on Software Engineering, ICSE 2022, Pittsburgh, PA, USA, May 25-27, 2022*. ACM, 1597–1608.

[Shi et al.(2022a)] Lin Shi, Fangwen Mu, Xiao Chen, Song Wang, Junjie Wang, Ye Yang, Ge Li, Xin Xia, and Qing Wang. 2022a. Are we building on the rock? on the importance of data preprocessing for code summarization. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE 2022, Singapore, Singapore, November 14-18, 2022*. ACM, 107–119.

[Vedantam et al.(2015)] Ramakrishna Vedantam, C. Lawrence Zitnick, and Devi Parikh. 2015. CIDEr: Consensus-based image description evaluation. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*. IEEE Computer Society, 4566–4575.

[Watson et al.(2020)] Cody Watson, Michele Tufano, Kevin Moran, Gabriele Bavota, and Denys Poshyvanyk. 2020. On learning meaningful assert statements for unit test cases. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*. 1398–1409.

[Wei et al.(2020)] Bolin Wei, Yongmin Li, Ge Li, Xin Xia, and Zhi Jin. 2020. Retrieve and Refine: Exemplar-based Neural Comment Generation. In *35th IEEE/ACM International Conference on Automated Software Engineering, ASE 2020, Melbourne, Australia, September 21-25, 2020*. IEEE, 349–360.

[Yu et al.(2022)] Hao Yu, Yiling Lou, Ke Sun, Dezhi Ran, Tao Xie, Dan Hao, Ying Li, Ge Li, and Qianxiang Wang. 2022. Automated Assertion Generation via Information Retrieval and Its Integration with Deep learning. In *44th IEEE/ACM 44th International Conference on Software Engineering, ICSE 2022, Pittsburgh, PA, USA, May 25-27, 2022*. ACM, 163–174.

[Zhang et al.(2020)] Jian Zhang, Xu Wang, Hongyu Zhang, Hailong Sun, and Xudong Liu. 2020. Retrieval-based neural source code summarization. In *ICSE '20: 42nd International Conference on Software Engineering, Seoul, South Korea, 27 June - 19 July, 2020*. ACM, 1385–1397.

[Zhou et al.(2019)] Yaqin Zhou, Shangqing Liu, Jing Kai Siow, Xiaoning Du, and Yang Liu. 2019. Devign: Effective Vulnerability Identification by Learning Comprehensive Program Semantics via Graph Neural Networks. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019*. 10197–10207.