南京郵電大學
Nanjing University of Posts and Telecommunications

Learning in the Wild: Towards Leveraging Unlabeled Data for Effectively Tuning Pre-trained Code Models

南京邮电大学

汇报人： 周伽芮

汇报时间：2025年6月6日

目录

研究背景

- **研究现状：**

深度学习技术的出现，尤其是预训练技术，在代码智能领域取得了显著进展。不同于传统的监督学习方法，这些预训练的代码模型首先使用自监督学习任务(如掩码语言建模MLM)在大规模未标记数据集上进行预训练，获取通用编程知识，然后在下游任务中对标记数据集进行微调。这种预训练的代码模型在代码总结、缺陷检测等代码智能任务上达到了最先进的性能。

但众所周知，深度学习模型需要大量数据，下游任务中标记数据集的大小对于预训练模型的性能很重要，下游任务中标记数据集的大小是通常由于**两个主要原因**受到限制：

1、数据集规模和质量有限

2、标注成本高昂

这个问题的一个可能的解决方案是在调优阶段通过**伪标记**来利用大量未标记的数据。

- **伪标记方法：**

利用有限标注数据训练teacher model，用以标注大量未标注数据集，生成伪标记数据。

将伪标记数据与原始标注数据合并，训练新的student model。可反复迭代提高模型性能。

该技术以任务特定的方式利用未标注数据，在图像分类、对话系统等任务上显示出良好前景。

研究目的

- **伪标记面临的挑战：**

伪标记数据质量参差不齐，存在噪声，直接使用可能会放大模型错误知识，降低性能。由于源代码复杂语义，识别和去除伪标记噪声并非易事。确保过滤后数据无噪声也是不现实的。

- **需要研究的方向：**

如何有效地利用有噪声的伪标记数据，使模型对代码智能任务具有容错性是至关重要的，但尚未得到充分的研究。

- **研究目的：**

在本文中提出了一种名为HINT的新方法，通过更好地利用伪标记数据来改进大规模未标记数据集的预训练代码模型。HINT包括两个主要模块：混合伪标记数据选择和耐噪训练。

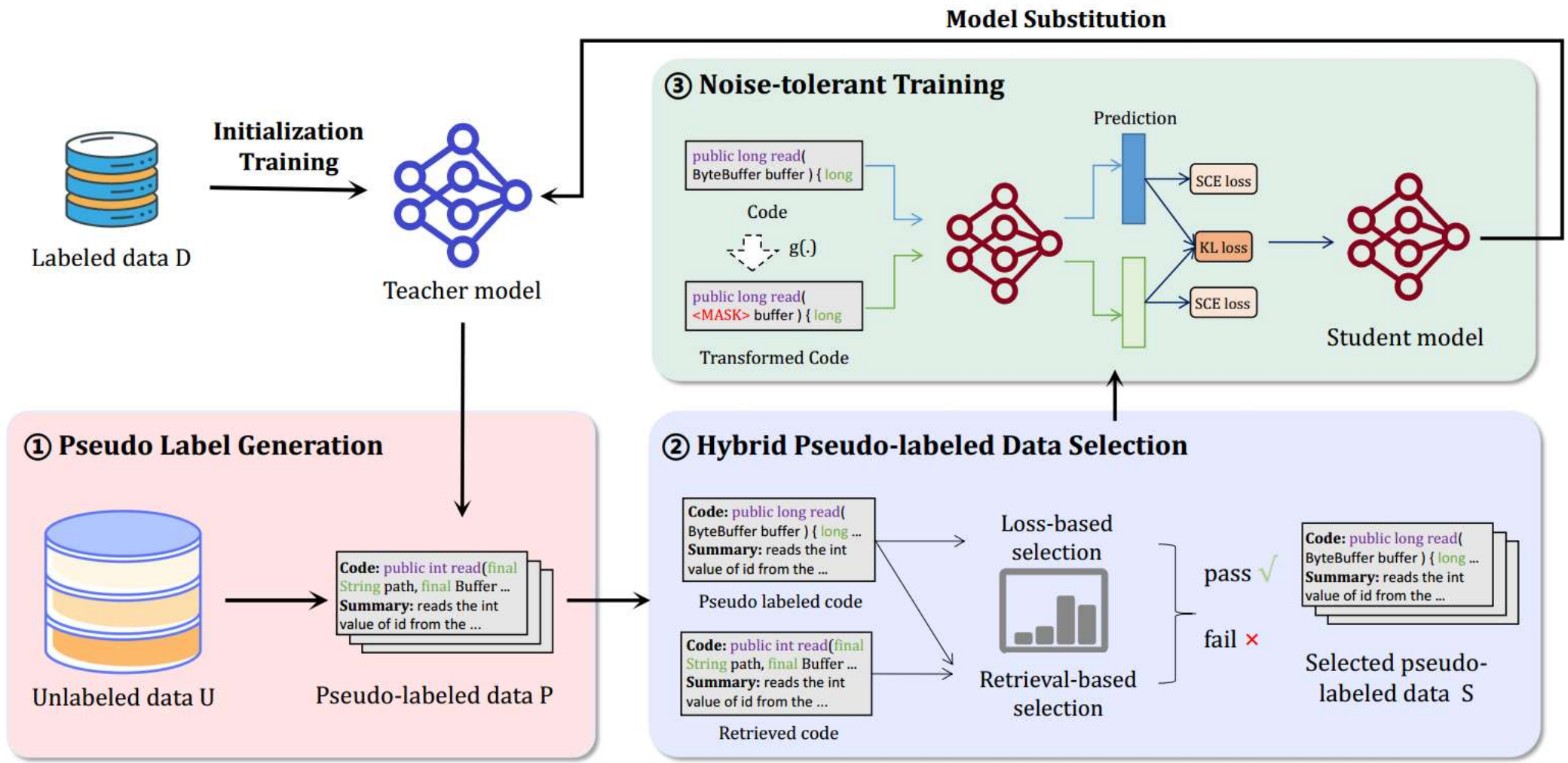


Figure 2: The overview of HINT.

研究方法

对于图1 (a)中的示例，我们可以观察到，尽管生成的总结的质量很差，但它的损失值很低。具体来说，当比较所有伪标记数据的损失时，它显示出比83%的伪标记数据更低的损失。

在图1 (b)中，生成的伪总结可以很好地描述Java代码段中检查两个对象是否等价的含义，但其训练损失值较高，超过了伪标记数据的52%。

```
#Sample from the unlabeled dataset
#Code
def validate_float(s)
    try:
        return float(s)
    except ValueError:
        raise ValueError('Could not convert %s to float' % s)

#Pseudo-labeled Summary
#converts a string to s s.
```

Low Training Loss

✗ Low quality

```
#Retrieved sample from the training dataset
#Code
def validate_int(s)
    try:
        return int(s)
    except ValueError:
        raise ValueError('Could not convert %s to int' % s)

#Summary
#convert s to int or raise.
```

Low similarity

(a) A Python example of low-quality pseudo-labeled data (top).

```
//Sample from the unlabeled dataset
//Code
static boolean areEqual(Object a, Object b){
   return (a==null?b==null:a.equals(b));
}
//Pseudo-labeled Summary
//check if two possibly null objects are equal.
```

High Training Loss

√ HIgh quality

```
//Retrieved sample from the training dataset
//Code
public static boolean equal(Object a, Object b){
   return a==b||(a!=null&&a.equals(b));
}
//Summary
//returns true if two possibly null objects are equal.
```

High similarity

(b) A Java example of high-quality pseudo-labeled data (top).

代码总结任务中的示例说明了混合伪标签数据选择方法的动机，表明仅基于损失的数据选择策略可能不正确地度量伪标签的质量。

研究方法

作者提出的基于检索的伪标记数据质量选择方法，主要包括以下步骤：

1、利用BM-25方法为每个未标注数据检索标注训练集中最相似的代码样本。
2、提出利用归一化编辑距离来衡量未标注数据及其伪标签与检索到的相似样本及其真实标签之间的相似度。
3、如果归一化编辑距离满足一定阈值条件，则将该样本视为高质量伪标记数据并保留；否则视为噪声数据并过滤掉。
4、对于无法通过基于检索方法判断的样本，结合teacher model计算的训练损失，选取损失值最低的前 K%样本作为补充。
5、最终得到由高质量伪标记数据和原始标注数据组成的训练集S，用于训练新的student model。
6、归一化编辑距离公式及判断标准阈值条件在算法1中给出。

$$NED(x,y) = \begin{cases} \frac{edit\_distance(x,y)}{||x||} & \textbf{if } x,y \in sequence \\ \mathrm{I}\{x \neq y\} & \textbf{if } x,y \in label \end{cases} \quad (1)$$

Algorithm 1 Algorithm of HINT

Input: labeled dataset D, unlabeled dataset U, threshold of edit distance t, the threshold in loss-based selection K, code transformation function g, iteration number I
Output: neural model $\mathcal{F}$

Train the teacher model $\mathcal{F}_t$ on D
for each i in I **do**
  Generate pseudo data P for U using $\mathcal{F}_t$
  Calculate the loss of samples in P using $\mathcal{F}_t$
  TK ← samples with the least top K% loss value in P
  $S \leftarrow \emptyset$
  for each sample $\{x_i, y_i'\}$ in P **do**
    Retrieve the most similar sample (x_j, y_j) from D
    if $NED(x_i, x_j) \leq t \wedge NED(y_i, y_j') \leq t$ **then**
      $S.insert(\{x_i, y_i'\})$
    else if $NED(x_i, x_j) \leq t \wedge NED(y_i, y_j') \geq 1 - t$ **then**
      continue
    else if $\mathcal{F}_t(x_i, y_i') \in$ TK **then**
      $S.insert(\{x_i, y_i'\})$
    end if
  end for
  $L \leftarrow D \cup S$
  Train the student model $\mathcal{F}_s$ with dataset L and transformation function g through Equation 4
  $\mathcal{F}_t \leftarrow \mathcal{F}_s$
end for
return model $\mathcal{F}_t$

01

$$l_{sce}(x,y) = -\sum_{c=1}^{C}(p(c|x_i)\log(q(c|x)) + q(c|x_i)\log(p(c|x))), \quad (2)$$

耐噪声损失函数:
传统交叉熵(CE)损失对有噪声数据敏感，模型容易被噪声数据误导。
作者提出使用对称交叉熵损失(SCE)，将CE损失与逆交叉熵(RCE)损失相结合。
RCE为所有样本分配相同权重，可减轻模型受小的标签噪声影响。

02

$$l_{cr} = KL(\mathcal{F}_s(x)||\mathcal{F}_s(ct(x))) + KL(\mathcal{F}_s(ct(x))||\mathcal{F}_s(x)), \quad (3)$$

一致性正则化:
考虑到伪标签可能有噪声和不可靠，增加了一致性正则化项。
对原始代码和经过转换(如掩码、替换等)的代码的预测强加一致性约束。
使用KL散度衡量原始代码和转换后代码的预测分布的差异作为正则化项。
这种一致性正则化可丰富监督信号，为student model提供更可靠的目标函数。

$$l = \sum_{(x,y)\in D\cup S}[\, l_{sce}(x,y) + l_{sce}(ct(x),y) + \mu \cdot l_{cr}\,], \quad (4)$$

03

最终损失函数:
将耐噪声损失SCE与一致性正则化损失相结合，在标注数据和伪标记数据上联合优化。
引入超参数平衡伪标签监督信号和一致性正则化的影响。

实验结果

我们通过解决以下四个研究问题来评估HINT:

RQ1: HINT可以为现有的预训练代码模型提供多少改进?

对RQ1的回答:HINT始终如一地在所有任务和指标上改进三个预训练的代码模型，表明它在为预训练的代码模型利用未标记数据方面的有效性。

RQ2:每个组件对HINT性能的影响是什么?

回答RQ2:混合伪标记数据选择模块和耐噪声训练模块中的所有组件对HINT的性能都有积极的影响。

RQ3: HINT在跨域场景中的表现如何?

对RQ3的回答:在目标域中不存在带注释的数据的跨域场景中，HINT可以大大提高预训练代码模型的性能。

RQ4:在不同的参数设置下，HINT的性能如何变化?

回答RQ4:不同的超参数设置会影响HINT在不同任务上的性能。我们的超参数设置获得了相对更好的结果。

实验结果

RQ1: HINT可以为现有的预训练代码模型提供多少改进?

即使只有一次迭代，HINT也可以显著提高不同现有预训练代码模型在所有数据集和指标上的性能。这表明HINT在利用未标记数据方面是有效的，并且有利于下游任务中与任务无关的预训练代码模型。

Table 1: Experimental results on code summarization. "*" denotes statistical significance in comparison to the base models (i.e., two-sided t-test with p-value< 0.01).

基准数据集

Approach		JCSD				PCSD			
		BLEU-4	ROUGE-L	METEOR	CIDEr	BLEU-4	ROUGE-L	METEOR	CIDEr
CodeBERT	Base model	13.30	26.75	8.10	0.58	17.94	32.35	9.79	0.59
	+HINT$_{(1)}$	14.58*	**29.06***	8.74*	0.69*	18.81*	34.18*	10.52*	0.69*
	+HINT$_{(5)}$	**14.64***	29.00*	**8.87***	**0.71***	**18.86***	**34.25***	**10.87***	**0.72***
CodeT5	Base model	16.67	34.28	11.39	1.05	21.13	40.27	15.69	1.22
	+HINT$_{(1)}$	18.32*	35.49*	**12.36***	1.22*	22.33*	41.42*	**16.31***	1.35*
	+HINT$_{(5)}$	**18.48***	**35.63***	12.29*	**1.24***	**22.55***	**41.67***	16.21*	**1.36***
UniXcoder	Base model	17.16	32.56	11.05	1.11	22.42	35.84	15.38	1.31
	+HINT$_{(1)}$	18.90*	35.16*	12.38*	1.28*	23.77*	41.67*	16.64*	1.48*
	+HINT$_{(5)}$	**19.79***	**35.83***	**13.12***	**1.36***	**23.98***	**41.93***	**16.83***	**1.50***

流行的指标

预训练代码模型

实验结果

RQ1: HINT可以为现有的预训练代码模型提供多少改进?

HINT可以帮助预训练的模型捕获易受攻击的代码片段的模式。此外，通过对比HINT(1)和HINT(5)的结果，我们还可以观察到，经过多次迭代，HINT可以获得更好的性能。

Table 2: Experimental results on defect detection. Statistical significance is not applicable to these metrics [10].

精确率　召回率　二者的调和平均值

Approach		Precision	Recall	F1
CodeBERT	Base model	29.64	17.63	22.11
	+HINT$_{(1)}$	30.81	21.52	25.34
	+HINT$_{(5)}$	**32.09**	**22.36**	**26.35**
CodeT5	Base model	31.38	20.32	24.66
	+HINT$_{(1)}$	36.79	22.36	27.81
	+HINT$_{(5)}$	**37.66**	**22.36**	**28.06**
UniXcoder	Base model	31.30	17.63	22.55
	+HINT$_{(1)}$	**33.28**	20.96	25.73
	+HINT$_{(5)}$	32.04	**22.26**	**26.27**

实验结果

RQ1: HINT可以为现有的预训练代码模型提供多少改进?

我们可以观察到HINT可以在很大程度上改进所有基线预训练模型。

具体而言，在CodeBERT上，HINT(5)在EM、LCS和ED方面分别提高了21.24%、5.29%和11.44%的基线。这表明更好地利用HINT的未标记数据的能力也有利于生成准确的断言语句。

Table 3: Experimental results on assertion generation. "*" denotes statistical significance in comparison to the base models (i.e., two-sided t-test with p-value< 0.01).

Approach		EM	LCS	ED
CodeBERT	Base model	31.82	65.99	21.68
	+HINT$_{(1)}$	37.75*	69.46*	**19.05***
	+HINT$_{(5)}$	**38.58***	**69.48***	19.20*
CodeT5	Base model	43.64	72.56	20.30
	+HINT$_{(1)}$	46.53*	74.32*	18.47*
	+HINT$_{(5)}$	**47.66***	**75.22***	**18.17***
UniXcoder	Base model	43.64	72.67	17.82
	+HINT$_{(1)}$	47.13*	74.72*	16.61*
	+HINT$_{(5)}$	**47.56***	**74.76***	**16.21***

EM、LCS越高、ED越低，表明模型生成的结果与参考更加接近。

对RQ1的回答:HINT始终如一地在所有任务和指标上改进三个预训练的代码模型，表明它在为预训练的代码模型利用未标记数据方面的有效性。

实验结果

RQ2:每个组件对HINT性能的影响是什么?

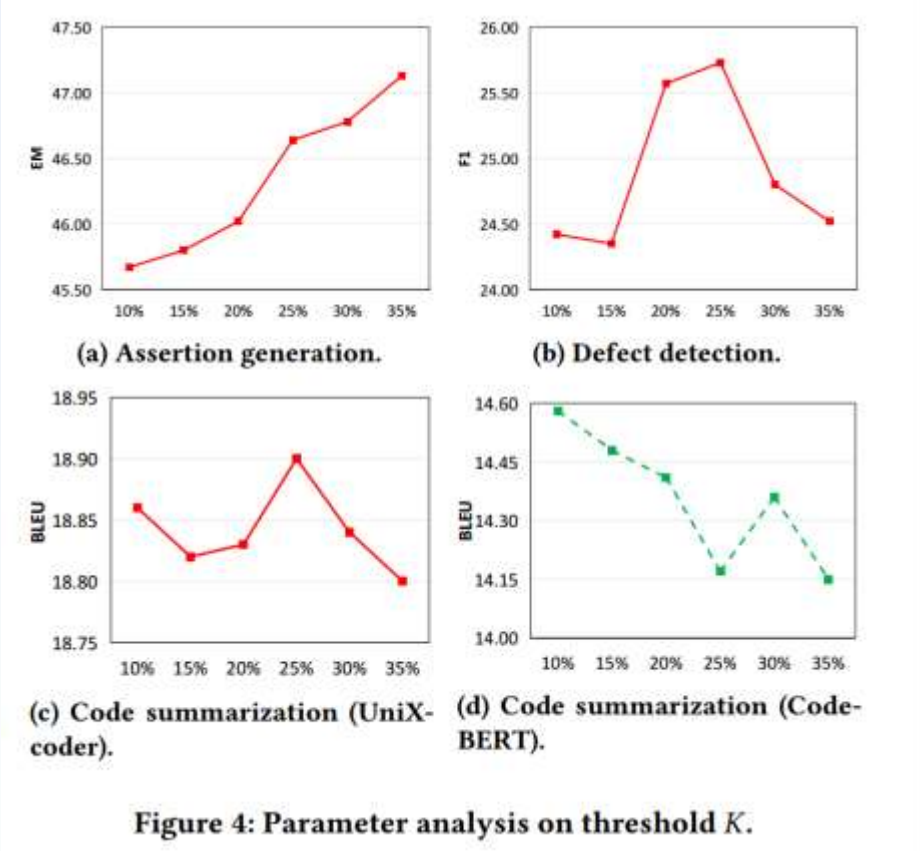
回答RQ2:混合伪标记数据选择模块和耐噪声训练模块中的所有组件对HINT的性能都有积极的影响。

Table 4: Ablation study of HINT. Best and second best results are marked in bold and underline respectively.

Approach	Code Summarization				Defect Detection			Assertion Generation		
	BLEU-4	ROUGE-L	METEOR	CIDEr	Precision	Recall	F1	EM	LCS	ED
UniXcoder+HINT	18.90	**35.16**	12.38	1.28	33.28	**20.96**	**25.73**	**47.13**	**74.72**	16.61
Random selection	18.34	34.11	11.70	1.23	34.39	16.14	21.97	45.27	73.65	17.10
-w/o loss-based selection	18.54	34.09	12.34	1.24	26.87	18.65	22.02	45.55	73.96	17.10
-w/o retrieval-based selection	18.71	34.91	12.21	1.26	32.33	19.94	24.67	45.03	73.50	17.16
-w/o data selection	18.27	34.44	12.03	1.21	**34.71**	16.42	22.29	47.03	74.64	16.65
-w/o noise tolerant loss	**18.93**	34.96	12.26	**1.29**	33.02	19.57	24.58	46.64	74.46	**16.56**
-w/o consistency regularization	18.78	35.03	**12.40**	1.27	33.82	19.29	24.57	46.81	74.55	16.70

实验结果

RQ4:在不同的参数设置下，HINT的性能如何变化?



(a) Assertion generation.

(b) Defect detection.

(c) Code summarization (UniXcoder).

(d) Code summarization (CodeBERT).

Figure 4: Parameter analysis on threshold K.

基于丢失数据选择的阈值K。我们通过实验来评估HINT在10%、15%、20%、25%、30%和35%不同阈值下的性能。K设置得越大，选择的伪标记样本就越多。如图4所示，在所有预训练的代码模型和任务上，随着K的增加，模型性能呈现出类似的趋势。HINT先增加并达到峰值，然后急剧下降，K增大。较大的K有涉及更多噪声数据的风险，而较小的K可能过于严格并过滤了许多高质量的样本。此外，我们可以发现，对于不同的模型和任务，K的最优值变化很大。

基于模型的性能越差，伪标记数据的质量越低。因此，当在不同的预训练代码模型上应用HINT时，可以在较强的基础模型上使用相对较大的K，反之亦然。

实验结果

RQ3: HINT在跨域场景中的表现如何?

如表5所示，HINT可以大大提高跨域场景中预训练代码模型的性能。其中，在Java to Python和Python to Java上，HINT分别使UniXcoder的BLEU-4分数提高了28.79%和30.96%，说明HINT可以有效地利用未标记数据中的知识。

Table 5: Experimental results on cross-domain scenario. "*" denotes statistical significance in comparison to the base models (i.e., two-sided t-test with p-value< 0.01).

Approach	Python → Java				Java → Python			
	BLEU-4	ROUGE-L	METEOR	CIDEr	BLEU-4	ROUGE-L	METEOR	CIDEr
CodeBERT	9.98	20.01	4.99	0.21	12.65	22.35	7.02	0.25
CodeBERT+HINT	**13.82***	**27.71***	**8.10***	**0.60***	**16.14***	**30.33***	**9.89***	**0.58***
CodeT5	7.75	12.55	7.12	0.29	14.81	30.59	9.66	0.84
CodeT5+HINT	**14.09***	**22.51***	**9.28***	**0.88***	**16.85***	**33.70***	**10.77***	**1.07***
UniXcoder	12.68	26.03	8.33	0.66	13.18	20.94	10.70	0.64
UniXcoder+HINT	**16.33***	**32.22***	**10.54***	**1.03***	**17.26***	**28.28***	**13.14***	**0.97***

对RQ3的回答:在目标域中不存在带注释的数据的跨域场景中，HINT可以大大提高预训练代码模型的性能。

实验结果

RQ4:在不同的参数设置下，HINT的性能如何变化?

t设置为0.4， λ=0.5时，HINT在缺陷检测和断言生成任务上取得了最佳性能。

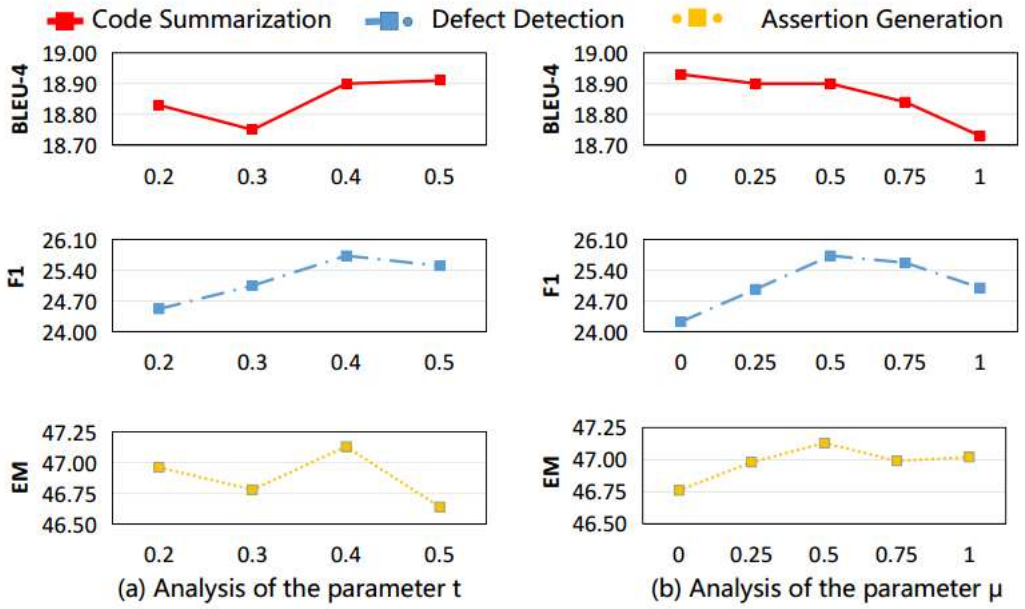


Figure 5: Parameter analysis on t and μ.

随着迭代次数增加，HINT的性能持续提升，大约在第5次迭代时达到峰值。这表明HINT能够充分利用未标注数据不断促进模型自我提升。

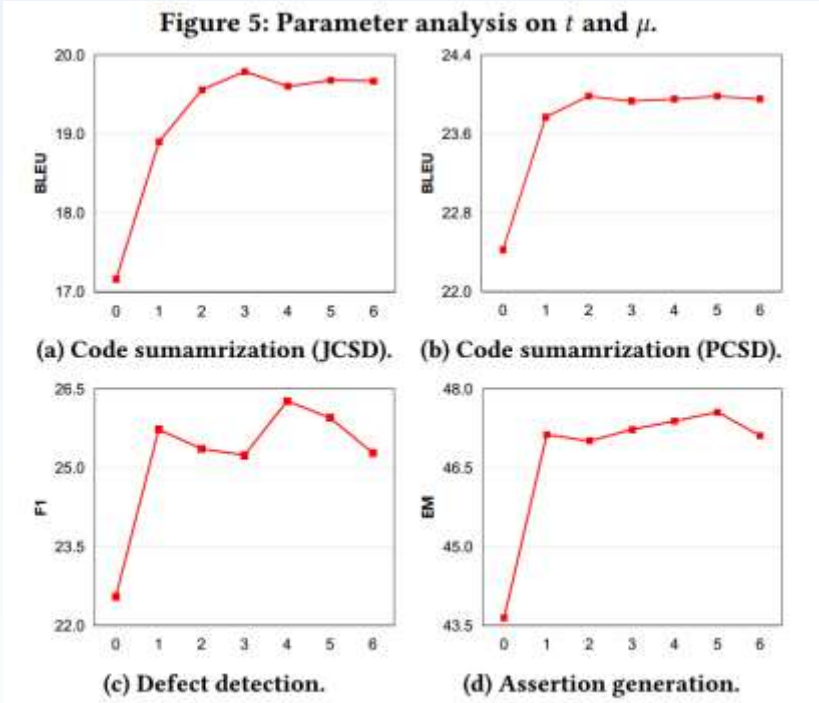


Figure 6: Performance on each Iteration.

回答RQ4:不同的超参数设置会影响HINT在不同任务上的性能。我们的超参数设置获得了相对更好的结果。

实验结果

是什么让HINT起作用?

1、利用伪标记数据改善预训练模型在下游任务的表现:

通过两个示例案例(代码总结和断言生成)，展示了HINT相比基线预训练模型(UniXcoder)能够生成更准确、更详细的预测结果。原因在于HINT能从大量类似模式的<代码片段，伪标签>对中学习，获取更丰富的知识。

2、能够选择质量更高的伪标记数据进行模型训练:

HINT的数据选择模块可以识别并过滤掉低质量的伪标记数据。通过计算伪标签与真实标签的平均编辑距离，发现HINT选择的数据集的距离明显小于未经过滤的所有伪标签。这表明HINT能够有效过滤噪声数据，选取高质量的伪标记数据用于模型训练。

```
//Code from the test set:
public Exchange(final Request request, final Origin origin){
    this.currentRequest = request;
    this.origin = origin;
    this.timestamp = System.currentTimeMillis();
}
```

Summary generated by Unixcoder:
constructs a new exchange object.
Summary generated by Unixcoder+HINT:
constructs an exchange with the given request and origin.
Ground truth summary:
creates a new exchange with the specified request and origin.

Code from the Unlabeled dataset:

```
public ScannerException(ErrorMessages message, int line){
    this(null, ErrorMessages.get(message), message, line, -_NUM);
}
```

Pseudo-labeled summary:
creates new scannerexception with message and line number.

Figure 7: Case study on the code summarization task. The green texts highlight the similar part between the prediction of UniXcoder+HINT and pseudo-labeled summary.

```
//Code from the test set:
hasSubscriptionId_emptyID(){
    when(fixture.getSubscriptionId()).thenReturn("");
    "<AssertPlaceHolder>";
}
hasSubscriptionId(){
    return
((getSubscriptionId())!=null)&&(!(getSubscriptionId().isEmpty()));
}
```

Assertion generated by Unixcoder:
org.junit.Assert.assertTrue(fixture.hasSubscriptionId());
Assertion generated by Unixcoder+HINT:
org.junit.Assert.assertFalse(fixture.hasSubscriptionId());
Ground truth assertion:
org.junit.Assert.assertFalse(fixture.hasSubscriptionId());

Code from the Unlabeled dataset:

```
hasNext_on_an_empty_collection__returns_false(){
    com.artemis.utils.IntBagIterator intBagIterator = new
com.artemis.utils.IntBagIterator(new com.artemis.utils.IntBag(99));
    "<AssertPlaceHolder>";
}
hasNext(){
    return  (this.cursor)<(size);
}
```

Pseudo-labeled assertion:
org.junit.Assert.assertFalse(intBagIterator.hasNext());

Figure 8: Case study on the assertion generation task. The red and green texts highlight the difference in predictions made by UniXcoder and UniXcoder+HINT.

讨论

- **HINT框架的两个主要局限性：**

  1、无法引入额外知识或纠正事实性知识错误

  2、依赖于基础模型的能力。

- **解决方法：**

  1、为了潜在地减轻这种限制，可以进一步研究通过与知识库或搜索引擎的交互将事实知识集成到预训练的代码模型中。

  2、将这种限制归因于模型能力的固有约束，并相信可以通过使用更先进的预训练代码模型来减轻这种限制。

总结

该论文工作的主要贡献如下：

(1)是第一个在代码智能任务的转弯阶段以特定任务的方式利用大规模未标记数据的方法。

(2)提出了一个新的框架HINT，利用大规模的未标记数据来有效地调整预训练的代码模型。

它首先以混合的方式选择高质量的伪标记数据，然后在训练过程中提高模型对噪声数据的容忍度。

(3)在三个任务上的大量实验表明，HINT可以建立在一系列现有的强预训练模型之上，并不断提高它们在许多下游任务上的性能。

下周科研规划：

1、尝试搭建环境，复刻实验内容。

2、阅读论文《Vul-RAG: Enhancing LLM-based Vulnerability Detection via Knowledge-level RAG》。

请老师批评指正