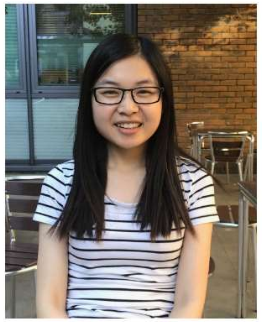
No More Fine-Tuning? An Experimental Evaluation of Prompt Tuning in Code Intelligence

2022 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2022)

Presenter: Zhu Jie

2022.8.26

Authors: HIT(SZ) + CUHK



Cuiyun Gao (C.Y.)

PhD, The Chinese University of Hong Kong, 2018

Tenured Associate Professor (校青年拔尖副教授)
School of Computer Science and Technology
Harbin Institute of Technolgy, Shenzhen
Email: gaocuiyun@hit.edu.cn

Expertise: Software repository mining & Natural language understanding »

Michael R. Lyu 呂榮聰

B.Sc. (National Taiwan University), M.Sc. (UC Santa Barbara), Ph.D. (UCLA)

- **IEEE Fellow (2004)**
- **AAAS Fellow (2007)**
- **Croucher Senior Research Fellow (2008)**
- **IEEE Reliability Society Engineer of the Year (2010)**
- **ACM Fellow (2015)**
- **China Computer Federation (CCF) Overseas Outstanding Contributions Award**
- **Choh-Ming Li Professor of Computer Science and Engineering (2020)**
- **The 13th Guanghua Engineering Science and Technology Award (2020)**

No More Fine-Tuning? An Experimental Evaluation of Prompt Tuning in Code Intelligence

Chaozheng Wang
Harbin Institute of Technology
Shenzhen, China
wangchaozheng@stu.hit.edu.cn

Yuanhang Yang
Harbin Institute of Technology
Shenzhen, China
ysngkil@gmail.com

Cuiyun Gao*
Harbin Institute of Technology
Shenzhen, China
gaocuiyun@hit.edu.cn

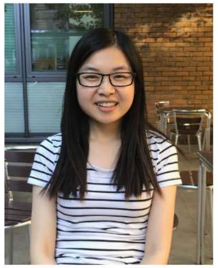
Yun Peng
The Chinese University of Hong Kong
Hong Kong, China
ypeng@cse.cuhk.edu.hk

Hongyu Zhang
The University of Newcastle
Newcastle, Australia
hongyu.zhang@newcastle.edu.au

Michael R. Lyu
The Chinese University of Hong Kong
Hong Kong, China
lyu@cse.cuhk.edu.hk

https://cuiyungao.github.io/
http://www.cse.cuhk.edu.hk/lyu/home

Authors: HIT(SZ) + CUHK



Cuiyun Gao (C.Y.)

PhD, The Chinese University of Hong Kong, 2018

Tenured Associate Professor (校青年拔尖副教授)
School of Computer Science and Technology
Harbin Institute of Technolgy, Shenzhen
Email: gaocuiyun@hit.edu.cn

Expertise: Software repository mining & Natural language understanding

Google Scholar DBLP

2022:

- TSE "Revisiting, Benchmarking and exploring API recommendation: How far are we?", Yun Peng, Shuqing Li, Wenwei Gu, Yichen Li, Wenxuan Wang, **Cuiyun Gao***, Michael Lyu. IEEE Transactions on Software Engineering (TSE), to appear.
- TR "Dynamically relative position encoding-based Transformer for automatic code edit", Shiyi Qi, Yaoxian Li, **Cuiyun Gao***, Xiaohong Su, Shuzheng Gao, Zibin Zheng, and Chuanyi Liu. IEEE Transactions on Reliability (TR), to appear.
- FSE "No more fine-tuning? An experimental evaluation of prompt tuning in code intelligence", Chaozheng Wang, Yuanhang Yang, **Cuiyun Gao***, Yun Peng, Hongyu Zhang, Michael R. Lyu. 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE), to appear.
- TOSEM "HINNPerf: Hierarchical interaction neural network for performance prediction of configurable systems", Jiezhu Cheng, **Cuiyun Gao**, Zibin Zheng. ACM Transactions on Software Engineering and Methodology (TOSEM), to appear.
- TOSEM "Code structure guided Transformer for source code summarization", Shuzheng Gao, **Cuiyun Gao***, Yulan He, Jichuan Zeng, Lun Yiu Nie, Xin Xia, Michael R. Lyu. ACM Transactions on Software Engineering and Methodology (TOSEM), to appear.
- SANER "Source code summarization with structural relative position guided Transformer", Zi Gong, **Cuiyun Gao***, Yasheng Wang, Wenchao Gu, Yun Peng and Zenglin Xu. The 29th IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER), 2022.
- Neural Networks "Enriching query semantics for code search with reinforcement learning", Chaozheng Wang, Zhenhao Nong, **Cuiyun Gao***, Zongjie Li, Jichuan Zeng, Zhenchang Xing, and Yang Liu, Neural Networks, to appear.
- IST "Understanding in-app advertising issues based on large scale app review analysis", **Cuiyun Gao**, Jichuan Zeng, David Lo, Xin Xia, Irwin King, Michael R Lyu. Information and Software Technology (IST)
- ICSE "Static inference meets deep learning: A hybrid type inference approach for Python", Yun Peng, **Cuiyun Gao***, Zongjie Li, Bowei Gao, David Lo, Qirun Zhang, and Michael Lyu. The 44th International Conference on Software Engineering (ICSE), 2022.

https://cuiyungao.github.io/
http://www.cse.cuhk.edu.hk/lyu/home

Why Choose This Paper?

- New: FSE'22 Paper
- Simple and Promising Method: Prompt Tuning
- Idea: The first paper using prompt tuning on code intelligence/SE
- Related Task: Code Summarization => Method Name Recommendation
- Open and Easy to Reproduce: Replication Package on GitHub

No More Fine-Tuning? An Experimental Evaluation of Prompt Tuning in Code Intelligence

Chaozheng Wang
Harbin Institute of Technology
Shenzhen, China
wangchaozheng@stu.hit.edu.cn

Yuanhang Yang
Harbin Institute of Technology
Shenzhen, China
ysngkil@gmail.com

Cuiyun Gao*
Harbin Institute of Technology
Shenzhen, China
gaocuiyun@hit.edu.cn

Yun Peng
The Chinese University of Hong Kong
Hong Kong, China
ypeng@cse.cuhk.edu.hk

Hongyu Zhang
The University of Newcastle
Newcastle, Australia
hongyu.zhang@newcastle.edu.au

Michael R. Lyu
The Chinese University of Hong Kong
Hong Kong, China
lyu@cse.cuhk.edu.hk

Replication Package Link: https://github.com/adf1178/PT4Code

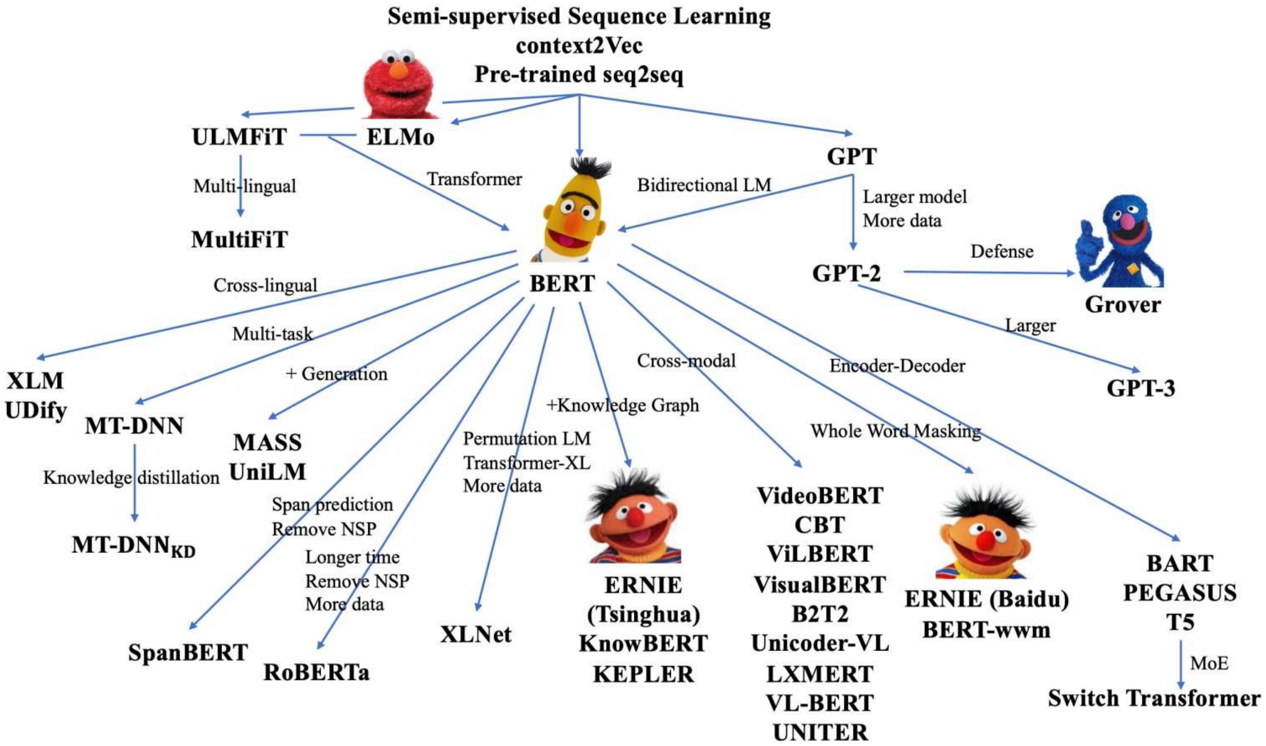
Background

Background: Code Intelligence

Please check "智能化代码开发的探索与展望 高翠芸.doc"

Background: Large Pre-trained Language Model

Pre-trained Language Models are Infrastructure in NLP



https://github.com/thunlp/OpenPrompt/tree/main/openprompt/prompts
https://www.bilibili.com/video/BV1UG411p7zv?p=56
https://www.openbmb.org/documentation/openprompt

The family of pre-trained language models (PLMs).

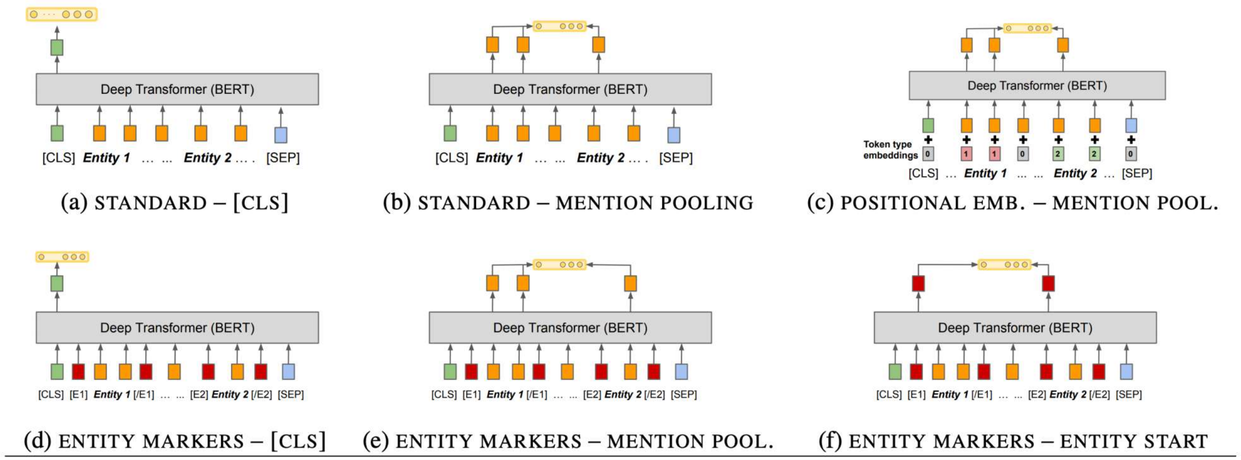
Background: Fine-tuning

Example Task: Relation Extraction using fine-tuning

- Extract the relation between two marked entities

LocatePlace

Tsinghua University is located in the northwest of Beijing



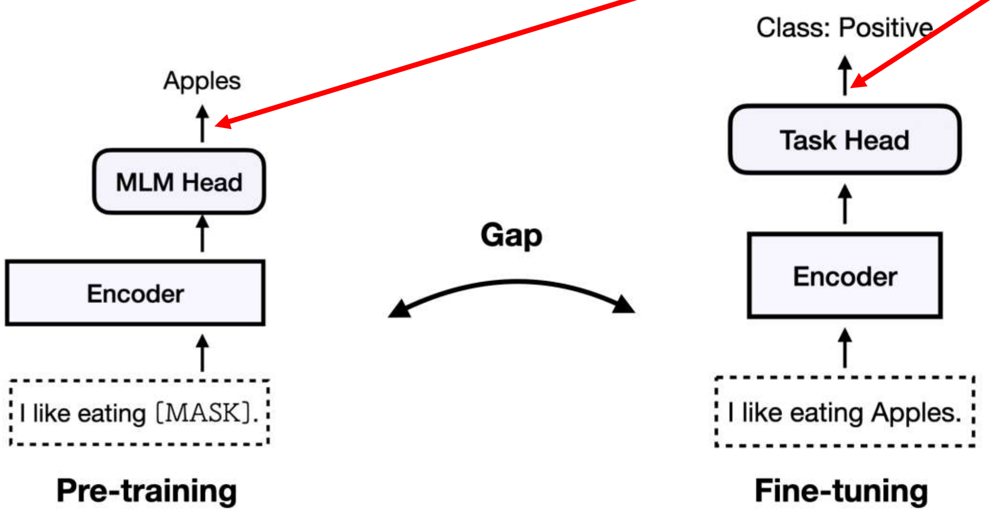
Matching the Blanks: Distributional Similarity for Relation Learning. 2019

https://github.com/thunlp/OpenPrompt/tree/main/openprompt/prompts
https://www.bilibili.com/video/BV1UG411p7zv?p=56
https://www.openbmb.org/documentation/openprompt

Background: Why prompt-tuning?

There is a gap between pre-training and fine-tuning

- Use PLMs as base encoders
- Add additional neural layers for specific tasks
- Tune all the parameters
- There is a GAP between pre-training and fine-tuning



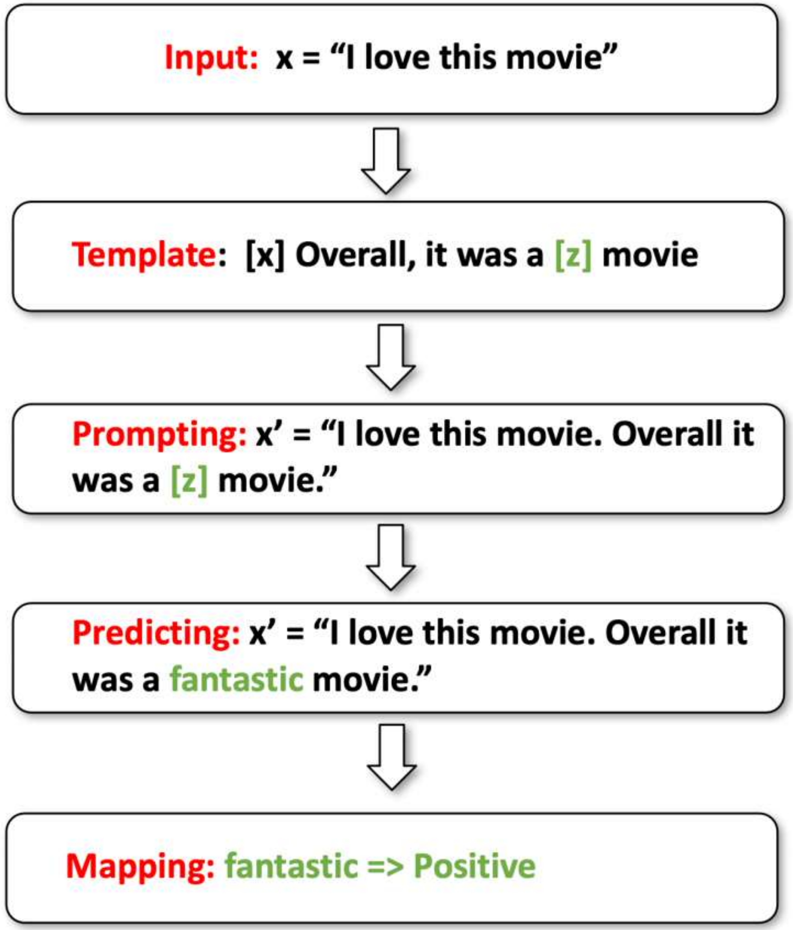
Gap: The inconsistent inputs and objectives between pre-training and fine-tuning render the knowledge of pre-trained models hard to be fully explored.

https://github.com/thunlp/OpenPrompt/tree/main/openprompt/prompts
https://www.bilibili.com/video/BV1UG411p7zv?p=56
https://www.openbmb.org/documentation/openprompt

Background: Prompt Tuning in NLP

Example Task: Sentiment Classification

- Sentiment Classification Task
  - Given a sentence, predict whether it is positive or negative
  - For example: “I hate this movie. it is terrible”. The sentiment classification model should give a negative prediction.
- Prompt Tuning/Learning
  - Build a template and fill it with the given input
  - Pass the template(with input) to the pretrained model
  - The model predicts the possibility of word distribution
  - Use verbalizer to map the output to the label(positive or negative)



https://github.com/thunlp/OpenPrompt/tree/main/openprompt/prompts
https://www.bilibili.com/video/BV1UG411p7zv?p=56
https://www.openbmb.org/documentation/openprompt

Background: Prompt Tuning in NLP

Type	Task	Input ([X])	Template	Answer ([Z])
Text CLS	Sentiment	I love this movie.	[X] The movie is [Z].	great fantastic ...
	Topics	He prompted the LM.	[X] The text is about [Z].	sports science ...
	Intention	What is taxi fare to Denver?	[X] The question is about [Z].	quantity city ...
Text-span CLS	Aspect Sentiment	Poor service but good food.	[X] What about service? [Z].	Bad Terrible ...
Text-pair CLS	NLI	[X1]: An old man with ... [X2]: A man walks ...	[X1]? [Z], [X2]	Yes No ...
Tagging	NER	[X1]: Mike went to Paris. [X2]: Paris	[X1][X2] is a [Z] entity.	organization location ...
Text Generation	Summarization	Las Vegas police ...	[X] TL;DR: [Z]	The victim ... A woman ... ...
	Translation	Je vous aime.	French: [X] English: [Z]	I love you. I fancy you. ...

Table 3: Examples of *input*, *template*, and *answer* for different tasks. In the **Type** column, "CLS" is an abbreviation for "classification". In the **Task** column, "NLI" and "NER" are abbreviations for "natural language inference" (Bowman et al., 2015) and "named entity recognition" (Tjong Kim Sang and De Meulder, 2003) respectively.

Pre-train, Prompt, and Predict: A Systematic Survey of Prompting Methods in Natural Language Processing

Background: Prompt Tuning in SE

Example: Use Prompt Tuning to Predict Defect (Yes or No)

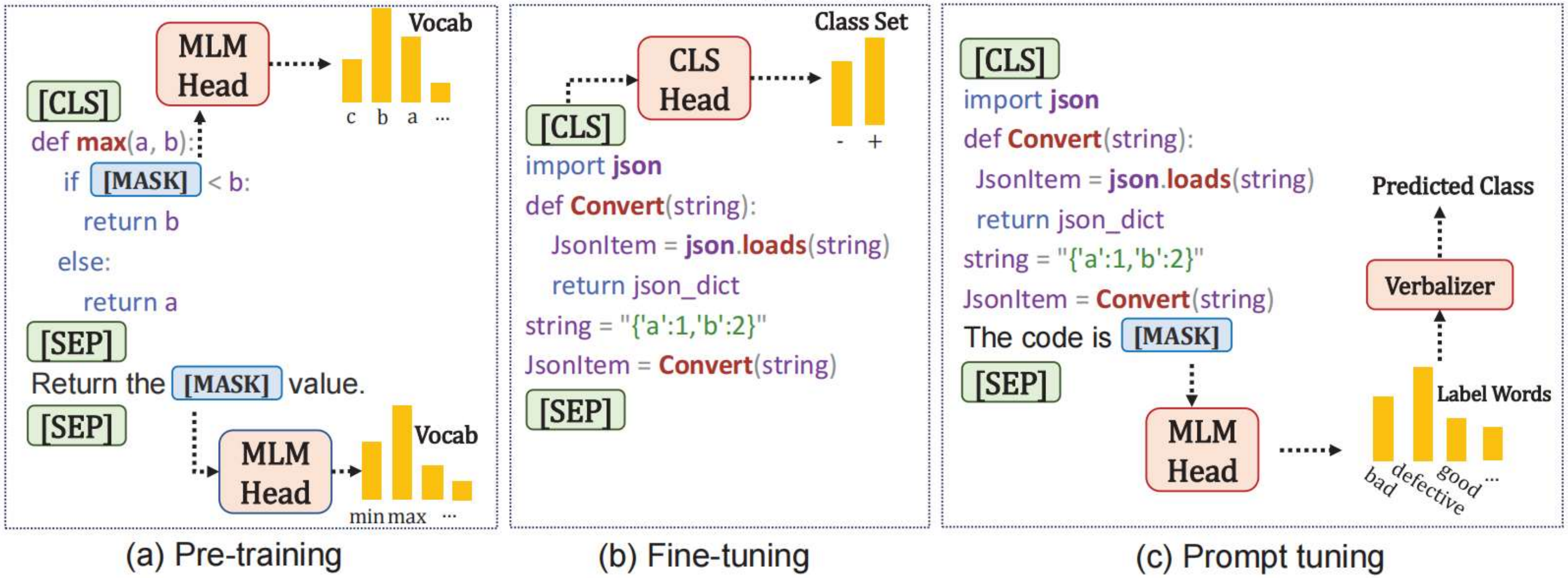


Figure 1: Illustration on the process of pre-training, fine-tuning, and prompt tuning on defect detection task. [*CLS*] **and** [*SEP*] **denote two special tokens in pre-trained models.**

Introduction/Logic

How to build A Solid Logic Chain

The state-of-the-art DL-based approaches to code intelligence exploit the pre-training and finetuning paradigm, in which language models are first pre-trained on a large unlabeled text corpora and then finetuned on downstream tasks. For instance: CodeBERT and CodeT5 ....

Conclusion 1:
These pre-trained source code models achieve significant improvement over previous approaches.

Introduction/Logic

How to build A Solid Logic Chain

Conclusion 2:

However, there exist gaps between the pre-training and finetuning process of these pre-trained models.

Pre-training models such as CodeBERT and CodeT5 are generally pre-trained using the Masked Language Modeling (MLM) objective. The input to MLM is a mixture of code snippets and natural language texts, and the models are trained to predict randomly masked input tokens. However, when models are fine-tuned into the downstream tasks, e.g. defect detection, the input involves only source code and the training objective changes to a classification problem.

The inconsistent inputs and objectives between pre-training and finetuning render the knowledge of pre-trained models hard to be fully explored, leading to sub-optimal results for downstream tasks. Besides, the performance of fine-tuning largely depends on the scale of downstream data.

Introduction/Logic

How to build A Solid Logic Chain

Conclusion 3:

Recently, prompt tuning is proposed to mitigate the above issues of fine-tuning.

By adding a prompt and verbalizer, prompt tuning reformulates the classification problem into an MLM problem, aligning the objective with the pre-training stage. This alignment unleashes the hidden power stored in the pre-trained models. Besides, the inserted natural language prompt can involve task-specific knowledge to facilitate the adaption to downstream tasks

Introduction/Logic

How to build A Solid Logic Chain

Conclusion 1:
These pre-trained source code models achieve significant improvement over previous approaches.

Conclusion 2:
However, there exist gaps between the pre-training and finetuning process of these pre-trained models.

Conclusion 3:
Recently, Prompt tuning is proposed to mitigate the above issues of fine-tuning.

Final Conclusion: No More Fine-Tuning!
Inspired by the success of prompt tuning in the NLP field, we would like to investigate if prompt tuning is effective for code intelligence tasks.

Experiment

Experiment PLM: CodeT5 and CodeBERT

Hugging Face | Search models, datasets, users... | Models | Datasets | Spaces | Docs | Solutions | Pricing | Log In | Sign Up

Salesforce/**codet5-base** | like 28

Text2Text Generation | PyTorch | Transformers | code_search_net | arxiv:2109.00859 | arxiv:1909.09436 | t5 | codet5 | AutoTrain Compatible | License: apache-2.0

Model card | Files and versions | Community | Train | Deploy | Use in Transformers

Edit model card

CodeT5 (base-sized model)

Pre-trained CodeT5 model. It was introduced in the paper CodeT5: Identifier-aware Unified Pre-trained Encoder-Decoder Models for Code Understanding and Generation by Yue Wang, Weishi Wang, Shafiq Joty, Steven C.H. Hoi and first released in this repository.

Disclaimer: The team releasing CodeT5 did not write a model card for this model so this model card has been written by the Hugging Face team (more specifically, nielsr).

Model description

From the abstract:

Downloads last month
13,826

Hosted inference API

Text2Text Generation

Inference API has been turned off for this model.

Dataset used to train Salesforce/codet5-base

code_search_net
Preview • Updated Jul 1 • 4.3k • 18

https://huggingface.co/Salesforce/codet5-base

Experiment: OpenPrompt

OpenBMB CPM-Live 工具包 模型 社区 文档 EN 登录 注册

OpenPrompt
v0.1.2

Search docs

GETTING STARTED
- Installation
- Introduction with an Example
- How to Write a Template?
  - Textual Template
  - Soft & Mix Template
  - Post processing
- How to Write a Verbalizer?
- FAQ

PACKAGE REFERENCE
- Base Classes
- Templates
- Verbalizer
- Prompt Generator
- Data Utils
- Data Processors
- Trainer
- Utils Functions
- Play with Configuration

Docs » How to Write a Template? View page source

How to Write a Template?

As we stated, template (which could be specific textual tokens or abstract new tokens, the only difference is the initialization) is one of the most important module in a prompt-leanring framework. In this tutorial, we introduce how to write a template and set the corresponding attributes for a `Template` class.

Our template language takes the insight fromthe Dict grammer from Python in order to make it easy-to-learn. We use a `meta` key to denote the orginal text input, or the part of the input, or other key information. A `mask` key is used to denote the indice of the token that need to be predicted. A `soft` key denotes soft tokens and textual tokens could be directly written down.

Textual Template

A simple template for binary sentiment classification, the `sentence` denotes the original input and the `mask` is the target position,

```
{"meta": "sentence"}. It is {"mask"}.
```

Here is a basic template for news topic classification, where one example contains two parts – a `title` and a `description`,

```
A {"mask"} news : {"meta": "title"} {"meta": "description"}
```

In entity typing, an `entity` is a key information, and we want to copy it in the template,s

```
{"meta": "sentence"} {"text": "In this sentence,"} {"meta": "entity"} {"text": "is a"} {"mask"},

# you can also omit the `text` key
{"meta": "sentence"}. In this sentence, {"meta": "entity"} is a {"mask"},
```

https://www.openbmb.org/documentation/openprompt

Experiment: Tasks and Datasets

- **Defect Detection**: The dataset is provided by Zhou et al. It contains 27K+ C code snippets from two open-source projects QEMU and FFmpeg, and 45.0% of the entries are defective.
- **Code Summarization**: We use the same dataset as the CodeT5 work. The dataset is from CodeSearchNet, which contains thousands of code snippet and natural language description pairs for six programming languages including Python, Java, JavaScript, Ruby, Go and PHP.
- **Code Translation:** The dataset is provided by Lu et al and is collected from four public repositories (including Lucene, POI, JGit and Antlr). Given a piece of Java (C#) code, the task is to translate the code into the corresponding C# (Java) version.

Table 1: Statistics of the datasets used in this paper.

Tasks	Datasets	Training Set	Val. Set	Test Set
Defect Detection	Defect	21,854	2,732	2,732
Code Summarization	Ruby	48,791	2,209	2,279
	JavaScript	123,889	8,253	6,483
	Go	317,832	14,242	14,291
	Python	409,230	22,906	22,104
	Java	454,451	15,053	26,717
	PHP	523,712	26,015	28,391
Code Translation	Translation	10,300	500	1,000

$$BP = \begin{cases} 1 & if\ c > r \\ e^{1-r/c} & if c \leq r \end{cases} \tag{9}$$

$$BLEU = BP \cdot exp\left(\sum_{n=1}^{N} w_n \log p_n\right) \tag{10}$$

where p_n means the modified n-gram precision and w_n is the weight. BP represents the brevity penalty, and c and r indicate the lengths of generated comment and target comment length, respectively. In our experiments, we choose smoothed BLEU-4 score, i.e., $n = 4$, for evaluating the generation tasks following previous work [9, 56].

Experiment: Prompt Design

Different Designs of Prompt Templates: Defect Prediction

$$f_{prompt}(x) = \text{“}The\ code\ [X]\ is\ [Z]\text{”}$$

$$\mathcal{V} = \begin{cases} + : & [defective, bad] \\ - : & [clean, perfect] \end{cases}$$

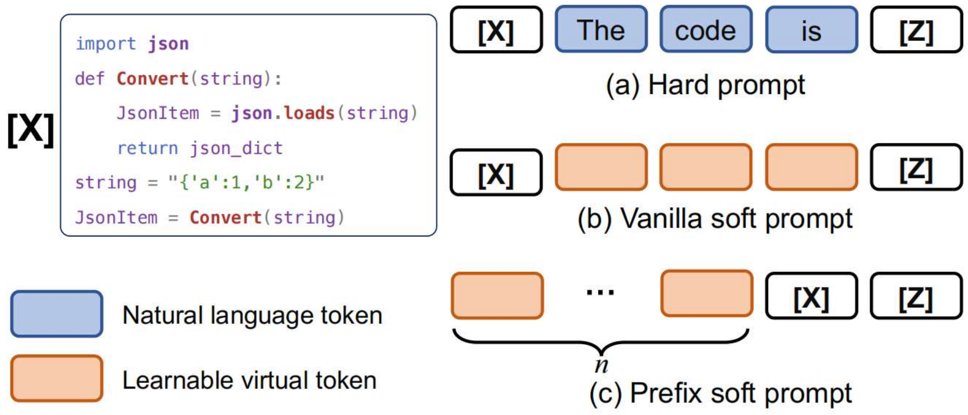


Figure 2: Illustration on the different types of prompt, where $[X]$ and $[Z]$ indicate the input slot and answer slot, respectively. Both vanilla soft prompt (b) and prefix soft prompt (c) belong to soft prompt.

Experiment: Prompt Design

Different Designs of Prompt Templates

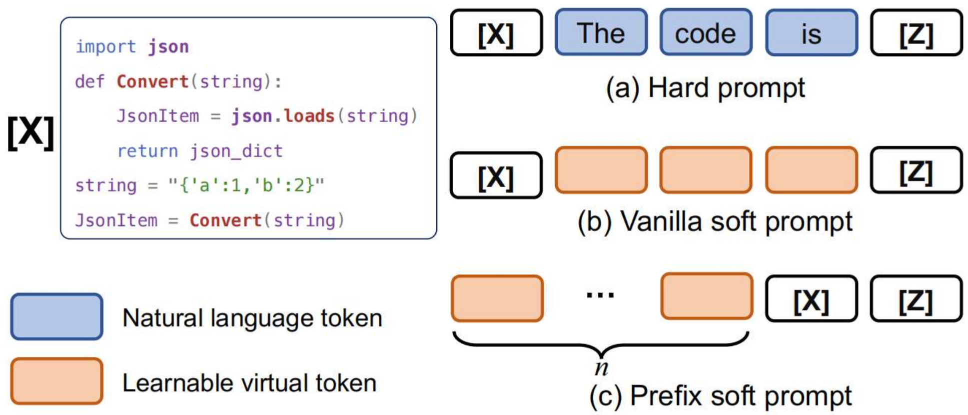


Figure 2: Illustration on the different types of prompt, where $[X]$ and $[Z]$ indicate the input slot and answer slot, respectively. Both vanilla soft prompt (b) and prefix soft prompt (c) belong to soft prompt.

Table 8: Classification accuracy (%) of comparing the performance of CodeBERT model on defect detection task via different prompt templates. The verbalizer is fixed as +: "*bad*", "*defective*"; −:"*perfect*", "*clean*". The underlined texts are replaced by virtual tokens in the corresponding vanilla soft prompt.

Hard Prompt	Vanilla Soft Prompt	Accuracy Hard	Accuracy Soft
$[X]$ The code is $[Z]$	$[X]$ $[SOFT] * 3$ $[Z]$	63.68	63.15
A $[Z]$ code $[X]$	$[SOFT]$ $[X]$ $[SOFT]$ $[Z]$	63.36	62.95
$[X]$ It is $[Z]$	$[X]$ $[SOFT][SOFT]$ $[Z]$	63.92	63.39
The code $[X]$ is $[Z]$	$[SOFT] * 2$ $[X]$ $[SOFT]$ $[Z]$	**64.17**	63.34

Table 9: Classification accuracy (%) of different verbalizers on the defect detect task, where the pre-trained model is CodeBERT. The template is "The code $[X]$ is $[Z]$".

Verbalizer	Accuracy
+: "Yes" −: "No"	63.08
+ : "bad" − : "perfect"	63.71
+ : "bad", "defective" − : "clean", "perfect"	64.17
+ : "bad", "defective", "insecure" − : "clean", "perfect", "secure"	63.26
+ : "bad", "defective", "insecure", "vulnerable − : "clean", "perfect", "secure", "invulnerable"	63.10

Experiment: Prompt Design

Prompt Templates on Code Summarization

```
# 加载CodeT5预训练模型
model_config = T5Config.from_pretrained("Salesforce/codet5-base")
plm = T5ForConditionalGeneration.from_pretrained("Salesforce/codet5-base", config=model_config)
tokenizer = RobertaTokenizer.from_pretrained("Salesforce/codet5-base")
WrapperClass = T5TokenizerWrapper

# 定义prompt模板
promptTemplate = SoftTemplate(model=plm, tokenizer=tokenizer, text='Code: {"placeholder":"text_a"} Summarization: {"mask"} ', initialize_from_vocab=True, num_tokens=50)

# 设置模型
model = PromptForGeneration(plm=plm, template=promptTemplate, freeze_plm=False, tokenizer=tokenizer, plm_eval_mode=False)
model.to(device)

# 训练
……….
```

Experiment Results: Code Summarization

Table 4: Results (BLEU-4 scores) of the CodeT5 model on code summarization task.

Methods		Ruby	JavaScript	Go	Python	Java	PHP	Overall
CodeT5-small	Fine-tuning	13.38	14.94	21.27	17.88	18.38	24.70	18.43
	Prompt tuning	**13.60**	**15.91**	**22.33**	**18.34**	**20.60**	**26.95**	**19.62**
CodeT5-base	Fine-tuning	13.70	15.80	22.60	17.97	19.56	25.77	19.23
	Prompt tuning	**14.29**	**16.04**	**23.11**	**18.52**	**19.72**	**27.06**	**19.79**

- **Statistically Better than Fine-tuning**: Moreover, prompt tuning can perform statistically better than fine-tuning at the significance level 0.05 on code summarization with a p-value 0.019.
- **Observe Consistent Improvement:** Compared with fine-tuning, prompt tuning obtains an improvement of 6.46% and 2.91% when using CodeT5-small and CodeT5-base as pre-trained models (6.46% = 1.19/18.43)
- **On Small PLMs:** The advantage of prompt tuning is more obvious for smaller pre-trained models.

Experiment Results: Other 2 tasks

Table 3: Classification accuracy on defect detection.

Methods		Accuracy
CodeBERT	Fine-tuning	62.12
	Prompt tuning	**64.17**
CodeT5-small	Fine-tuning	62.96
	Prompt tuning	**63.91**
CodeT5-base	Fine-tuning	65.00
	Prompt tuning	**65.82**

Table 5: Experimental results on code translation tasks: Java-C# and C#-Java.

Methods		C# to Java			Java to C#		
		BLEU	Accuracy	CodeBLEU	BLEU	Accuracy	CodeBLEU
CodeT5-small	Fine-tuning	78.67	65.40	82.55	82.29	63.80	87.01
	Prompt tuning	**79.59**	**66.00**	**83.06**	**83.33**	**64.30**	**87.99**
CodeT5-base	Fine-tuning	79.45	**66.10**	83.96	83.61	65.30	88.32
	Prompt tuning	**79.76**	**66.10**	**84.39**	**83.99**	**65.40**	**88.74**

RQ1: How effective is the prompt tuning in solving code intelligence tasks?

- **Finding 1:** Prompt tuning is more effective than fine-tuning on the code intelligence tasks, with respect to different pre-trained models and different programming languages. Besides, the advantage of prompt tuning is more obvious for smaller pre-trained models.

Experiment: Data Scarcity Scenarios

RQ2: How capable is prompt tuning to handle data scarcity scenarios?

- **Experiment Setting 1:** low-resource scenario, in which there are significantly few training instances
- **Experiment Setting 2:** cross-domain scenario, in which the model is trained on a similar data-sufficient domain and tested on target domain. (transfer learning)

Table 6: Classification accuracy (%) on defect detection in low-resource scenario. '-' denotes the model fails to converge due to extreme lack of training data.

Method		Zero shot	16 shots	32 shots	64 shots	128 shots
CodeBERT	Fine-tuning	50.52	52.15	53.01	53.61	55.28
	Prompt tuning	**53.99**	**52.98**	**53.83**	**54.28**	**56.19**
CodeT5-small	Fine-tuning	-	-	51.22	52.10	54.28
	Prompt tuning	-	-	**52.36**	**53.59**	**55.04**
CodeT5-base	Fine-tuning	-	-	51.25	52.64	54.52
	Prompt tuning	-	-	**52.44**	**53.82**	**55.47**

To avoid randomness in data selection, we produce each subset five times with different seeds and run four times on each dataset. The average results are reported.

Experiment: Data Scarcity Scenarios

RQ2: How capable is prompt tuning to handle data scarcity scenarios?

- **Experiment Setting 1:** low-resource scenario, in which there are significantly few training instances

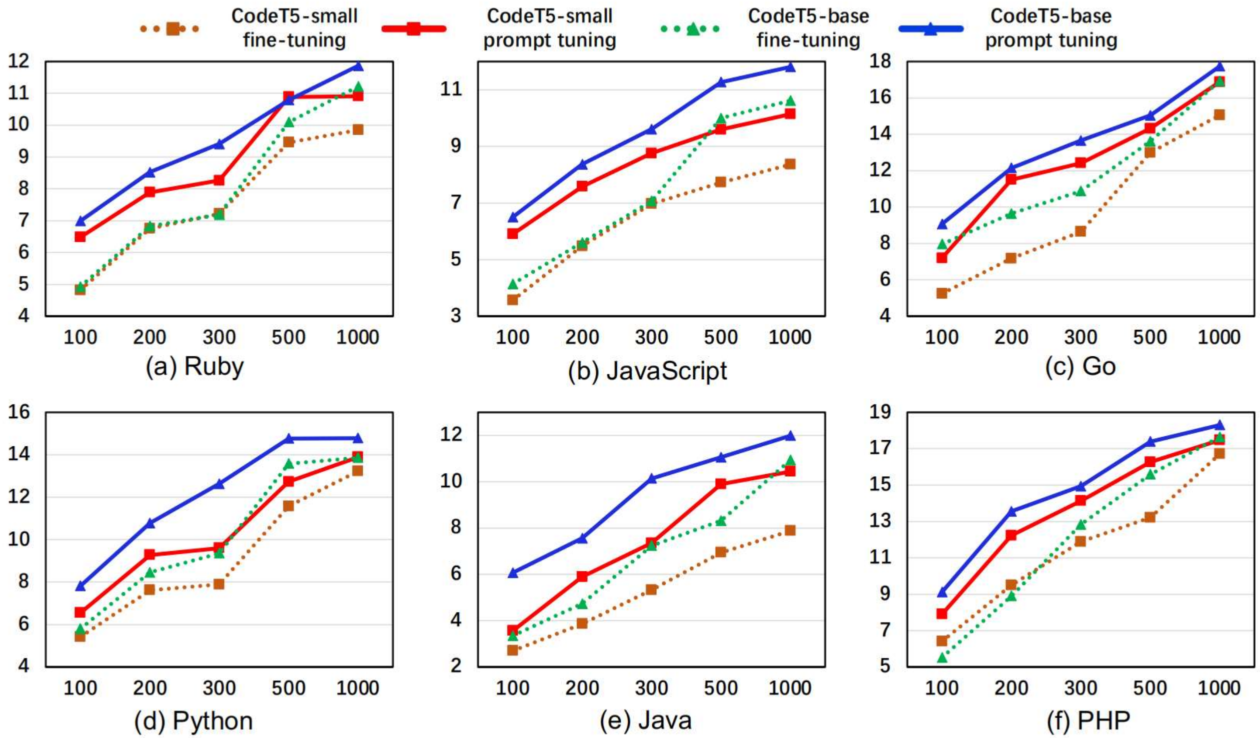


Figure 4: Results of fine-tuning and prompt tuning on code summarization task in low resource scenarios. The horizontal axis indicates the number of training instances while the vertical axis means the BLEU-4 score.

Experiment: Data Scarcity Scenarios

RQ2: How capable is prompt tuning to handle data scarcity scenarios?

- **Experiment Setting 2:** cross-domain scenario, in which the model is trained on a similar data-sufficient domain and tested on target domain.
- The data sizes of languages such as Java and Python are greatly larger than those of languages including Javascript and Ruby
  - Transfer learning: Transferring the knowledge of similar domains with sufficient data to the target domains
- We perform training on the programming language Java or Python, and evaluate on the language with fewer data such as Ruby, JavaScript, and Go.

> - **Finding 2:** Prompt tuning is more effective in low-resource scenarios than fine-tuning. The fewer training instances, the larger the improvement achieved by prompt tuning. Besides, prompt tuning also shows superior performance on the cross-domain code intelligence task.

Table 7: Experimental results (BLEU-4 score) on cross-language code summarization. The models are trained on Python or Java datasets, and tested on Ruby, JavaScript and Go, respectively.

Training	Methods	Ruby	JavaScript	Go
CodeT5-small				
Python	Fine-tuning	12.75	**12.37**	11.57
	Prompt tuning	**13.01**	12.35	**12.15**
Java	Fine-tuning	12.20	11.45	10.96
	Prompt tuning	**12.59**	**11.84**	**11.15**
CodeT5-base				
Python	Fine-tuning	13.06	12.81	12.89
	Prompt tuning	**13.37**	**13.11**	**14.27**
Java	Fine-tuning	12.67	11.50	11.88
	Prompt tuning	**13.13**	**11.99**	**12.96**

Experiment: Data Scarcity Scenarios

RQ3: How different prompt templates affect the performance of prompt tuning?

1. hard prompt template (manually defined)
2. hard prompt v.s. vanilla soft prompt;
3. length of prefix soft prompt.

Table 10: Results (BLEU-4 scores) of prompt tuning with different prompt templates on the code summarization task. There is no verbalizer for the prompts of generation tasks.

	$f_{prompt}(\cdot)$	Ruby	JavaScript	Go	Python	Java	PHP	Overall
CodeT5-small	Summarize [LANG] [X] [Z]	13.45	15.01	21.20	17.82	18.43	24.52	18.41
	[SOFT] * 2 [X] [Z]	13.33	14.96	21.17	17.93	18.29	24.61	18.38
	Generate comments for [LANG] [X] [Z]	13.44	14.96	21.24	17.90	18.52	24.46	18.42
	[SOFT] * 4 [X] [Z]	13.49	14.87	21.29	17.92	18.34	24.68	18.44
CodeT5-base	Summarize [LANG] [X] [Z]	13.67	15.91	22.51	18.00	19.63	25.76	19.25
	[SOFT] * 2 [X] [Z]	13.86	15.75	22.48	18.12	19.52	25.91	19.27
	Generate comments for [LANG] [X] [Z]	13.68	15.84	22.49	18.03	19.59	25.88	19.25
	[SOFT] * 4 [X] [Z]	13.74	15.82	22.63	18.06	19.60	25.83	19.28

- **Finding 3:** Template design for hard prompt is more important for the classification task than the generation task. Too short or long lengths of prefix prompts can degrade the model performance. Hard prompts present better prediction accuracy than the corresponding vanilla soft prompts.

Table 8: Classification accuracy (%) of comparing the performance of CodeBERT model on defect detection task via different prompt templates. The verbalizer is fixed as +: "*bad*", "*defective*"; −:"*perfect*", "*clean*". The underlined texts are replaced by virtual tokens in the corresponding vanilla soft prompt.

Hard Prompt	Vanilla Soft Prompt	Accuracy Hard	Accuracy Soft
[X] The code is [Z]	[X] [SOFT] * 3 [Z]	63.68	63.15
A [Z] code [X]	[SOFT] [X] [SOFT] [Z]	63.36	62.95
[X] It is [Z]	[X] [SOFT][SOFT] [Z]	63.92	63.39
The code [X] is [Z]	[SOFT] * 2 [X] [SOFT] [Z]	**64.17**	63.34

Table 9: Classification accuracy (%) of different verbalizers on the defect detect task, where the pre-trained model is CodeBERT. The template is "The code [X] is [Z]".

Verbalizer	Accuracy
+: "Yes" −: "No"	63.08
+: "bad" −: "perfect"	63.71
+: "bad", "defective" −: "clean", "perfect"	64.17
+: "bad", "defective", "insecure" −: "clean", "perfect", "secure"	63.26
+: "bad", "defective", "insecure", "vulnerable −: "clean", "perfect", "secure", "invulnerable"	63.10

Experiment: Case Study

```
Turntabler.AuthorizedUser.update_laptop", "original_string": "def update_laptop(name)
      assert_valid_values(name, *%w(mac pc linux chrome iphone cake intel android))
      api('user.modify', :laptop => name)
      self.attributes = {'laptop' => name}
      true
    end", "language": "ruby", "code": "def update_laptop(name)
      assert_valid_values(name, *%w(mac pc linux chrome iphone cake intel android))
      api('user.modify', :laptop => name)
      self.attributes = {'laptop' => name}
      true
    end
```

(a) Ground truth comment: Updates the laptop currently being used

(b) Comment generated by fine-tuning: Modify the laptop.

(c) Comment generated by prompt tuning: Update the laptop.

Figure 6: Case study on the code summarization task, where the pre-trained model is CodeT5-small.

```
public virtual bool
contains(object o){
    return indexOf(o) != -1;
}
```

(a) Original C# code

```
public boolean contains(Object o) {
    return indexOf(o) != -1;
}
```

(b) Ground truth Java code

```
public boolean contains(Object o)
{
    return indexOf(o);
}
```

(c) Generated Java code by fine-tuning

```
public boolean contains(Object o) {
    return indexOf(o) != -1;
}
```

(d) Generated Java code by prompt tuning

Figure 7: Case study on the code translation task, where the pre-trained model is CodeT5-small.

Future Directions

- **Consider more characteristics of source code:** like syntactic structures, in the design of template and the choices of verbalizer. Domain knowledge plays an important role on the design of prompts.
  - How to better utilize "Code Structure Information/Code Context"

```
InputExample(
    guid = i,
    text_a = code_tokens[i] + ' Identifier Names: ' + identifiers[i],
    text_b = classes[i][:-2] + ' File Name: ' + file_names[i][:-2] + ', Method Signature: ' +
    parse_comment(signatures[i]),
    tgt_text = names[i],
)
```

- **Interpretability and Robustness:** through constructing cloze-style prompt template, the factual knowledge and biases contained in the pre-trained models can be investigated. Researchers can focus on improving the interpretability and robustness of pre-trained models and designing novel pre-training tasks in the future

Future Directions from CNSoft

智能化代码开发 – 展望

- 更好地融入结构信息的代码表示学习
  - 代码表示学习是提升下游任务的表现的关键
  - 针对代码的神经网络架构
- 代码中的预训练模型
  - CodeBERT,GraphCodeBERT等已在多个方面展示了他们强大的能力
  - 如何设计出更好的预训练任务
- 低资源场景下的代码语义学习
  - 一些早期语言如COBOL，缺少足够的训练数据，却仍在一些工业设施和政府系统中使用
  - 在few-shot，zero-shot场景下的探究

Future Directions from CNSoft

智能化代码开发 - 展望

- 生成类任务更合理的评价指标
  - 许多研究表明现有的指标可能无法很好地反应生成序列的质量
  - 以注释生成为例，如何衡量代码和注释的功能一致性，减少生成错误的注释

- 代码与自然语言之间的语义鸿沟
  - 如何更好地对齐代码和自然语言之间的语义

- 模型的可解释性
  - 给出做出判断的依据

Threats to Validity

- **Limited Datasets:** To mitigate this issue, we choose the **most widely-used datasets for each code-related task**, modify the seeds and run the sampling multiple times. We also plan to collect more datasets in the future to better evaluate the
- **Limited downstream tasks:** Our experiments are conducted on three code intelligence tasks, including one classification task and two generation tasks. Although these tasks are the representative ones in code intelligence, there are many other tasks, such as code search and bug fixing. We believe that **we could obtain similar observations on these tasks since they can all be formulated as either classification tasks or generation tasks** for source code. We will evaluate more tasks with prompt tuning in our future work.
- **Suboptimal prompt design:** We demonstrate that prompt tuning can improve the performance of pre-trained models. However, the prompts we use in this paper may not be the best ones. It is **challenging to design the best prompt templates and verbalizers**, which will be an interesting future work.

Why is this paper accepted?

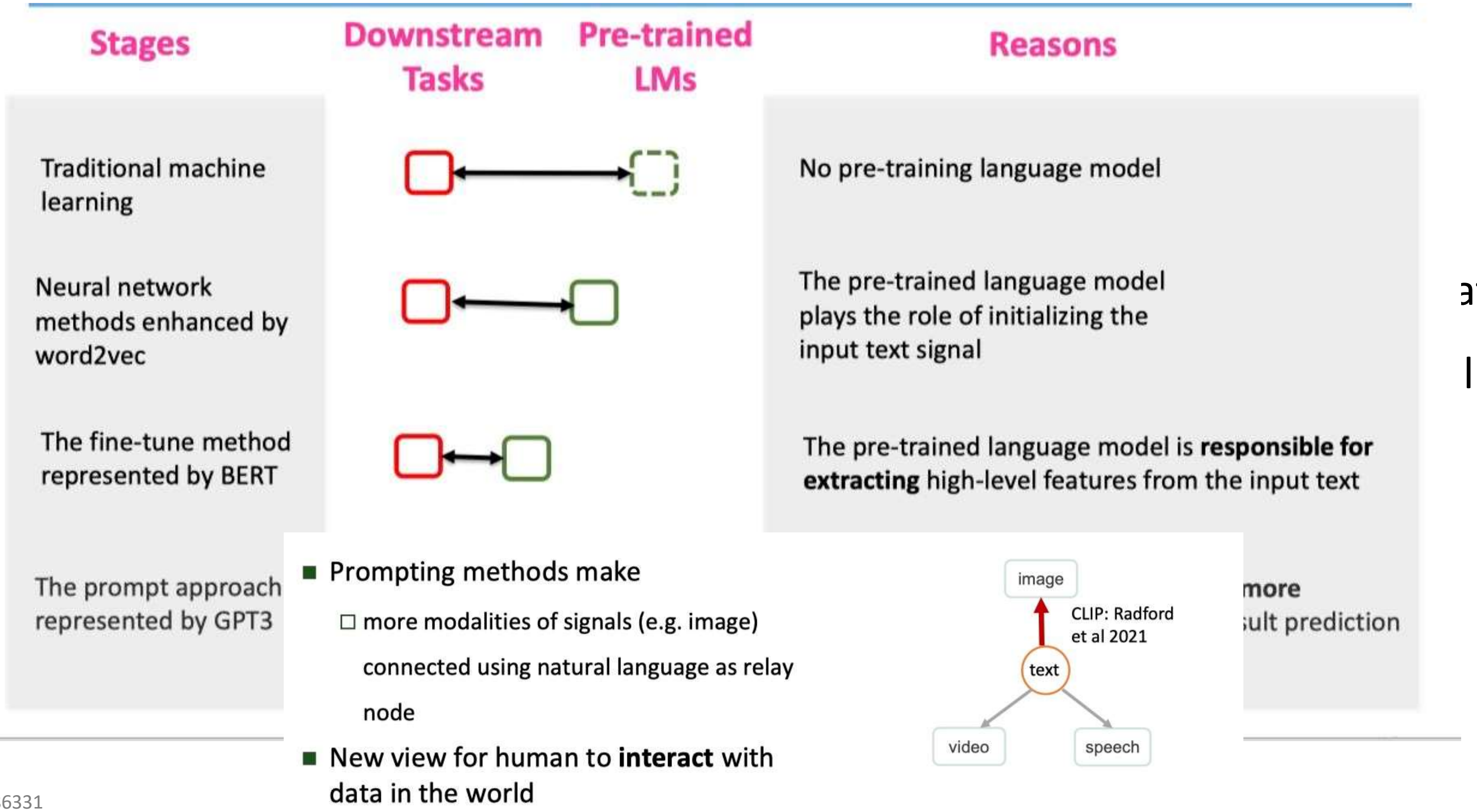
- **Sufficient experiment work: > 200 experiments** (3 tasks x 6 languages x 2 PLMs x 3 prompts, **4*V100 128GB**)
- **Prospective study:** the first paper using prompt tuning (remains unexplored)
- Solid Logic Chain: Introduction part of the paper
- Convincing Baselines: Compared with fine-tuning/transformer etc.

Code translation

		BLEU	Accuracy	CodeBLEU	BLEU	Accuracy	CodeBLEU
	Naive copy	18.69	0	-	18.54	0	-
	Transformer	50.47	37.90	61.59	55.84	33.00	63.74
	RoBERTa (code)	71.99	57.90	80.18	77.46	56.10	83.07
	CodeBERT	72.14	58.00	79.41	79.92	59.00	85.10
CodeT5-small	Fine-tuning	78.67	65.40	82.55	82.29	63.80	87.01
CodeT5-small	Prompt tuning	**79.59**	**66.00**	**83.06**	**83.33**	**64.30**	**87.99**
CodeT5-base	Fine-tuning	79.45	66.10	83.96	83.61	65.30	88.32
CodeT5-base	Prompt tuning	**79.76**	**66.10**	**84.39**	**83.99**	**65.40**	**88.74**

Reflection: Advantages of prompt

- A new and universal paradigm of NLP: close the gap between pre-training and fine-tuning
- Outperfor
- A better w
  - We ca
  - For ex ate as:
  - “Code ller: xxxxx, …….”
  - We ne



Stages	Downstream Tasks	Pre-trained LMs	Reasons
Traditional machine learning			No pre-training language model
Neural network methods enhanced by word2vec			The pre-trained language model plays the role of initializing the input text signal
The fine-tune method represented by BERT			The pre-trained language model is **responsible for extracting** high-level features from the input text
The prompt approach represented by GPT3			**more** ult prediction

https://zhuanlan.zhihu.com/p/442486331
https://zhuanlan.zhihu.com/p/399295895
https://www.bilibili.com/video/BV1Sf4y1g7ra

Reflection: Criticisms of prompt

- 感觉是旧瓶装新酒啊。。。现代deep learning就是为了规避feature engineering，可是到了prompt这里选择template和answer不还是feature engineering嘛。
- prompt是个好的研究方向，但目前实际用处确实不大。如果固定预训练参数，可以减小模型储存空间 + 训练速度加快 + 小样本效果小幅提升，但样本变多后效果就差于全量finetune。暂时只对非常特定的场景有帮助。
- 感觉在预训练模型本身尚有多种问题存在的前提下，在预训练过程与prompt本身脱节的前提下，追求利用prompt以摆脱finetuning似不现实，所以感觉感觉prompt更像是某种探针，用来探测模型学到哪些则尚可，用来进行下游任务可能值得商榷。
- Hard Prompt难以设计，AutoPrompt效率低下，SoftPrompt对模型和数据要求高，verbalizer设计也很麻烦。

https://zhuanlan.zhihu.com/p/442486331
https://zhuanlan.zhihu.com/p/399295895
https://www.bilibili.com/video/BV1Sf4y1g7ra

Reflection: preprocessing tricks

- Filter by Length: too long/short seq is abnormal (>128, <3)
- Subword/Split words: getResponese => get response
- Lowercase: GET => get
- Concat Single Chars: CONSTANT_NameAndType => constant name and type
- Remove/Preserve punctuation
- AST Structure: javalang, tree-sitter

```python
programtokens=javalang.tokenizer.tokenize(programtext)
res = ["{} {}".format(token.__class__.__name__, token.value) for token in programtokens]
print(programtext[:106]+'\n')
print(', '.join(res[:20]))
```

✓ 0.3s Python

```
public static int BubbleSortFloat2(float[] num) {
    int last_exchange;
    int right_border = num.length
```

```
Modifier public, Modifier static, BasicType int, Identifier BubbleSortFloat2, Separator (, BasicType float,
Separator [, Separator ], Identifier num, Separator ), Separator {, BasicType int, Identifier last_exchange,
Separator ;, BasicType int, Identifier right_border, Operator =, Identifier num, Separator ., Identifier length
```

Reflection: AST

```
MethodDeclaration(annotations=[], body=[LocalVariableDeclaration(annotations=[], declarators=
[VariableDeclarator(dimensions=[], initializer=None, name=last_exchange)], modifiers=set(),
type=BasicType(dimensions=[], name=int)), LocalVariableDeclaration(annotations=[], declarators=
[VariableDeclarator(dimensions=[], initializer=BinaryOperation(operandl=MemberReference(member=length,
postfix_operators=[], prefix_operators=[], qualifier=num, selectors=[]), operandr=Literal(postfix_operators=[],
prefix_operators=[], qualifier=None, selectors=[], value=1), operator=-), name=right_border)], modifiers=set(),
type=BasicType(dimensions=[], name=int)), DoStatement(body=BlockStatement(label=None, statements=
[StatementExpression(expression=Assignment(expressionl=MemberReference(member=last_exchange, postfix_operators=[],
prefix_operators=[], qualifier=, selectors=[]), type==, value=Literal(postfix_operators=[], prefix_operators=[],
qualifier=None, selectors=[], value=0)), label=None), ForStatement(body=BlockStatement(label=None, statements=
[IfStatement(condition=BinaryOperation(operandl=MemberReference(member=num, postfix_operators=[], prefix_operators=
[], qualifier=, selectors=[ArraySelector(index=MemberReference(member=j, postfix_operators=[], prefix_operators=[],
qualifier=, selectors=[]))]), operandr=MemberReference(member=num, postfix_operators=[], prefix_operators=[],
qualifier=, selectors=[ArraySelector(index=BinaryOperation(operandl=MemberReference(member=j, postfix_operators=[],
prefix_operators=[], qualifier=, selectors=[]), operandr=Literal(postfix_operators=[], prefix_operators=[],
qualifier=None, selectors=[], value=1), operator=+))]), operator=>), else_statement=None, label=None,
then_statement=BlockStatement(label=None, statements=[LocalVariableDeclaration(annotations=[], declarators=
[VariableDeclarator(dimensions=[], initializer=MemberReference(member=num, postfix_operators=[], prefix_operators=
[], qualifier=, selectors=[ArraySelector(index=MemberReference(member=j, postfix_operators=[], prefix_operators=[],
qualifier=, selectors=[]))]), name=temp)], modifiers=set(), type=BasicType(dimensions=[], name=float)),
StatementExpression(expression=Assignment(expressionl=MemberReference(member=num, postfix_operators=[],
prefix_operators=[], qualifier=, selectors=[ArraySelector(index=MemberReference(member=j, postfix_operators=[],
prefix_operators=[], qualifier=, selectors=[]))]), type==, value=MemberReference(member=num, postfix_operators=[],
prefix_operators=[], qualifier=, selectors=[ArraySelector(index=BinaryOperation(operandl=MemberReference(member=j,
postfix_operators=[], prefix_operators=[], qualifier=, selectors=[]), operandr=Literal(postfix_operators=[],
prefix_operators=[], qualifier=None, selectors=[], value=1), operator=+))])), label=None),
StatementExpression(expression=Assignment(expressionl=MemberReference(member=num, postfix_operators=[],
prefix_operators=[], qualifier=, selectors=[ArraySelector(index=BinaryOperation(operandl=MemberReference(member=j,
postfix_operators=[], prefix_operators=[], qualifier=, selectors=[]), operandr=Literal(postfix_operators=[],
prefix_operators=[], qualifier=None, selectors=[], value=1), operator=+))]), type==,
```

?

Conclusion

- Keywords/SVM/TF-IDF => RNN/LSTM/GRU => Transformer => BERT => Fine-tuning => Prompt Tuning?
- If it works in code summarization, it should also work in other areas, such as method name prediction.
- Prompt tuning might provide us with a new approach to combine and utilize different contexts.
  - Use domain knowledge(downstream task characteristics) to help design prompt templates.
- To make improvements / further study
  - Data Cleaning/Preprocessing: Avoid “Garbage In, Garbage Out”, what about the quality of training corpus? Could we make some rules/tools to standardize them first?
  - Prompt Ensemble
  - How to discover and utilize more contexts

Better Methods, More Contexts, Better Utilization

Thanks

Zhu Jie

2022.08.26