Don't Complete It! Preventing Unhelpful Code Completion for Productive and Sustainable Neural Code Completion Systems

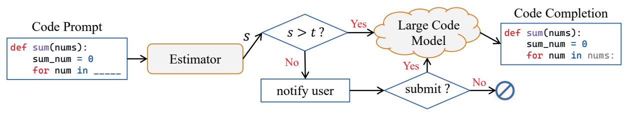
Zhensu Sun*, Xiaoning Du†, Fu Song*, Shangwen Wang‡, Mingze Ni§, Li Li†
* ShanghaiTech University, Shanghai, China
† Monash University, Melbourne, Australia
‡ National University of Defense Technology, Changsha, China
§ University of Technology Sydney, Sydney, Australia
{sunzhs, songfu}@shanghaitech.edu.cn, {xiaoning.du, li.li}@monash.edu,
wangshangwen13@nudt.edu.cn, Mingze.Ni@student.uts.edu.au

***Abstract*—Currently, large pre-trained language models are widely applied in neural code completion systems. Though large code models significantly outperform their smaller counterparts, around 70% of displayed code completions from Copilot are not accepted by developers. Being reviewed but not accepted, their help to developer productivity is considerably limited. Even worse, considering the high cost of the large code models, it is a huge waste of computing resources and energy. To fill this significant gap, we propose an early-rejection mechanism to turn down low-return prompts by foretelling the code completion qualities without sending them to the code completion system. Furthermore, we propose a lightweight Transformer-based estimator to demonstrate the feasibility of the mechanism. The experimental results show that the proposed estimator helps save 23.3% of computational cost measured in floating-point operations for the code completion systems, and 80.2% of rejected prompts lead to unhelpful completion.**

I. Introduction

Motivated by the unprecedented superior performance, a large number of commercial applications based on the pre-trained Large Code Models (LCMs) are recently released. Aiming for seamless assistance to developers, code completion systems are designed to automatically and actively issue code completion requests to the servers when a typing phase is detected. However, not all the code completions generated by the state-of-the-art code completion systems are helpful to the developers. For instance, according to the study of Ziegler et al. [1] where they surveyed developers' feedback on Copilot code completions and 2,631 responses were collected, around 70% of displayed code completions are *not* accepted by developers. Undoubtedly, such a large proportion of unhelpful code completions pose significant threats to the main objective of code completion systems – to improve developer productivity, and put relevant operation costs in vain.

To reduce unhelpful code completions, we propose to equip the LCM-based code completion systems with an early-rejection mechanism to turn down low-return prompts that lead to unhelpful completions based on quality estimation before completion (named QEBC). We estimate the quality of completions since it is a quantifiable indicator of the abstract



```
def sum(nums):
    sum_num = 0
    for num in _____
```

```
def sum(nums):
    sum_num = 0
    for num in nums:
```

Fig. 1: The workflow of an LCM-based code completion system with QEBC.

concept—— helpfulness, where low-quality completions are more likely to be unhelpful. As shown in Figure 1, the core idea of QEBC is to have a lightweight code completion quality estimator for the code completion system, which guards the prompts sent to the LCM. It foretells the completion qualities of the LCM solely based on prompts. Whenever the estimated completion quality is lower than a chosen threshold, it blocks the completion request. Besides, we propose a lightweight Transformer-based completion quality estimator (TCQE) , trained with prompts and their completion quality indicators from the LCMs. We mainly aim to demonstrate the feasibility of QEBC, and test how well TCQE can fulfill the goals of QEBC.

In the experiment, a user study is conducted to investigate the developer productivity of LCMs with/without TCQE, where only 5.5% of the potential low-return prompts predicated by TCQE are annotated as accepted in our human study when 5% of overall prompts are predicated as potential low-return prompts by TCQE. More importantly, given a code prompt, it only costs 1.9 Giga Floating-point Operations (GFLOPs) for TCQE to yield an estimation score, while GPT-2 and CodeGen take 112.2 and 182.0 GFLOPs respectively to generate the completion (containing 10 tokens).

To the best of our knowledge, we are the first to propose a general and effective mechanism to prevent unhelpful code completions in a cost-friendly manner for LCM-based code completion systems. This work sheds light on the serious energy waste problem in neural code completion and opens a new research direction in this field. The source code are available on our website [2].

II. Proposed Method

A. Quality Estimation Before Completion

To address the problem, we propose to have a lightweight completion quality estimator which foretells how well a prompt can be completed by an LCM without executing the actual inference and call the mechanism QEBC (Quality Estimation Before Completion). Prompts will be turned down without getting completed if the completion quality is estimated to be lower than a pre-defined threshold. The strictness of QEBC can be flexibly controlled by adjusting the threshold. To be specific, we define QEBC as follows:

Definition 1
Given an LCM-based code completion system M and a code completion prompt p, a completion quality estimator E will estimate the quality of the completions to be generated by M for p, i.e., before p is actually processed. If the estimated score s is lower than a pre-defined threshold t, p will be blocked from being sent to the LCM-based code completion system M.

B. Transformer-based Completion Quality Estimator

Considering the outstanding performance of Deep Learning (DL) in semantic understanding of code, we seek to design a lightweight DL model to learn the mapping between the prompts (in the training dataset) and the quality of their completions from a target LCM, which can be seen as a regression task. Thus, we propose a Transformer-based completion quality estimator (TCQE) to fulfill this task. It is a Transformer backbone with a linear head layer that maps the hidden state of the backbone to the estimated accuracy score. Considering the cost of TCQE, we limit the number of trainable parameters in TCQE to 16 million. The training dataset for TCQE consists of pairs of a prompt and the accuracy score of its completion generated by the target LCM. Specifically, we split each code snippet C of a code dataset at a random position into two parts, where the former part C_x works as a prompt and the latter part serves as its ground truth completion C_y. Further, we run the target LCM M for the prompt C_x to obtain its completion prediction C'_y and calculate its accuracy against the ground truth C_y with respect to an accuracy metric as the score s, such as BLEU. Finally, by pairing each prompt C_x with its score s, we obtain a code-score dataset, with which we train the TCQE model.

III. Experiments

The goal of our experiments is to evaluate the quality of the target LCM with TCQE and the cost-friendliness of TCQE, which further demonstrates the feasibility of QEBC. We evaluate TCQE with a popular LCMs from our research community, GPT-2, on a code dataset, COFIC (Java).

A. Quality of completions

We apply the LCM to generate completions for code prompts. Specifically, we have five groups of completions under different settings, where each group is generated by GPT-2 using the Java testing dataset but filtered by various estimators: one RAND estimator (give random scores to every prompt), two TCQE estimators (respectively set to reject 5% and 25% prompts and trained with the BLEU metric), and two COC estimators (use the length of the prompt as the estimation score and respectively reject 5% and 25% prompts). We conduct a human study to annotate the acceptance of the completions of both rejected prompts and retained prompts in each group. The results are reported in Table I, indicating TCQE can effectively block unhelpful completions for developers, thus boosting developer productivity in practice.

TABLE I: The acceptance rate of the completions generated by the LCM with TCQE and other baseline estimators.

Estimator	**Acceptance Rate**			
	Bottom 5%		**Bottom 25%**	
	Retained	**Rejected**	**Retained**	**Rejected**
RAND	43.6%			
COC	45.2%	8.9%	44.6%	35.0%
TCQE	**46.7%**	**5.5%**	**54.3%**	**19.8%**

TABLE II: Scale and inference cost of TCQE and other baseline LCMs. m is the number of tokens in a piece of code completion generated by the LCM.

Models	**Param.**	**GFLOPs**			**Time (s)**		
		$m=10$	$m=20$	$m=50$	$m=10$	$m=20$	$m=50$
TCQE	16 M	**1.9**			**0.02**		
GPT-2	124 M	112.2	165.4	238.1	0.30	0.59	1.49
CodeGen	357 M	182.0	194.7	223.0	0.82	1.68	4.24

B. Cost-friendliness of TCQE

In this experiment, we evaluate the cost-friendliness of TCQE and compare it with the two LCMs, i.e., GPT-2 and CodeGen. The cost-friendliness is measured by the computational cost of LCMs and TCQE for handling a prompt containing 256 tokens in the same environment, where LCMs generate a piece of code completion (respectively 10, 20, and 50 new tokens are generated) while TCQE predicts an estimation score. We record the total number of floating point operations (represented by GFLOPs) and running time during the code completion or score prediction as the indicator of computational cost, using the Flops Profiler of DeepSpeed [3]. The scale and computing cost of TCQE and each LCM are reported in Table II. Though the figure may not be precise considering the cost of other factors such as communications, it suffices to demonstrate the cost-friendliness of TCQE, which satisfies our expectation for a feasible estimator.

References

[1] A. Ziegler, E. Kalliamvakou, S. Simister, G. Sittampalam, A. Li, A. S. Rice, D. Rifkin, and E. Aftandilian, "Productivity assessment of neural code completion," *ArXiv*, vol. abs/2205.06537, 2022.
[2] (2023) Qebc. [Online]. Available: https://sites.google.com/view/qebc/
[3] (2022) microsoft/deepspeed: Deepspeed is a deep learning optimization library that makes distributed training and inference easy, efficient, and effective. [Online]. Available: https://github.com/microsoft/DeepSpeed