Enhancing Natural Language to Code Generation in the SantaCoder Model through In-Context Learning

Kapu Nirmal Joshua[1] and Mihit Sreejith[2]

[1]Department of Electrical Engineering, IIT Kanpur
[2]Department of Computer Science, IIT Guwahati
[3]IBM AI Research, Bangalore

June 18, 2024

Abstract

Generating executable code from natural language instructions using Large Language Models (LLMs) presents challenges such as semantic understanding and handling ambiguous input. This study focuses on the SantaCoder model and explores the impact of in-context learning on code generation using the MBPP and HumanEval datasets for evaluation. Our results demonstrate significant improvements in three key metrics (defined in the paper): *correctness@k, similarity@k and pass@k*. To address the problem of selecting optimal demonstrations to maximize correctness and pass rates, we investigated two methods: latent concept selection and random selection. These findings highlight the effectiveness of in-context learning and the critical role of demonstration selection in enhancing the accuracy, efficiency, and versatility of the SantaCoder model in code generation.

Contents

1 Introduction

The problem of generating code from natural language using Large Language Models (LLMs) involves creating systems capable of translating human language instructions into executable code accurately. This requires the LLM to understand the semantics of the natural language input, grasp the intent behind the instructions, and convert it into syntactically correct and functional code in a specified programming language. Key challenges include handling ambiguous or imprecise language, ensuring the generated code is both correct and efficient, and covering a wide range of programming scenarios and languages.

A promising approach to solving this problem is in-context learning, where the LLM is provided with examples of natural language instructions paired with their corresponding code snippets as part of the input. By analyzing these examples, the model learns to map new, unseen instructions to the appropriate code outputs without explicit retraining. In-context learning allows the model to adapt to specific tasks by updating the context with relevant examples, thus providing a flexible and efficient method for generating code. This approach leverages the model's existing knowledge and pattern recognition capabilities, enabling it to interpret and execute new instructions accurately, making it a valuable tool for enhancing productivity in software development.

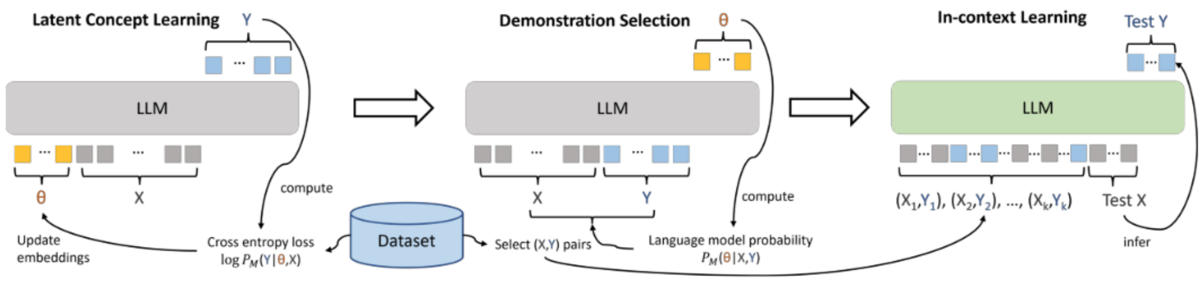

Figure 1: Few Shot Learning Pipeline For Code Generation with Latent Concept Learning

Further, we tested the effects of in-context learning using the SantaCoder model on the MBPP and HumanEval datasets and saw a significant increase in three key parameters: pass@k, correctness@k and @k. To determine the optimal demonstrations for increasing correctness and pass rates, we explored two selection methods: latent concept selection and random based selection. Our findings underscore the importance of demonstration selection in maximizing the effectiveness of in-context learning for code generation. All our code is available at `https://github.com/amarazad/ICL4Code`.

2 Methodology

In this section, we outline the methodology employed to integrate in-context learning into our code generation pipeline. We discuss four crucial steps taken to enhance the capabilities of our system. Firstly, we delve into the process of latent concept learning, where the model acquires implicit knowledge from provided examples. Subsequently, we elaborate on our approach to demonstration selection, which is crucial to optimizing the model learning process. Following this, we detail the methods employed for output formatting to ensure that the generated code adheres to the syntactic and semantic correctness. Finally, we examine our strategy for code evaluation, essential for assessing the quality and performance of the generated code.

2.1 Latent Concept Learning

In latent concept learning, denoted by the task-specific latent parameter θ_d, the objective is to imbue the model with task-specific knowledge encapsulated within a set of new token embeddings, termed as concept tokens. Initially, θ_d resides within a latent space, disconnected from the model's existing token embeddings. To integrate θ_d into the model's framework, a process known as soft prompt tuning is employed. This involves concatenating the input token embeddings with the concept tokens, represented by trainable tensors optimized via backpropagation. Our explanation is inspired by the generation process of a topic model, i.e. a simple latent variable model:

$$P(w_{1:T}) = \int_{\Theta} P(w_{1:T}|\theta)P(\theta)d\theta \tag{1}$$

where

$$\theta \in \Theta \tag{2}$$

Θ is the space of the topic/concept variable θ, and $w_{1:T}$ refers to the token sequence of a piece of text. Note that the topic model here refers to the modern neural topic models. On the other hand, generative LLMs model text data according to the general probabilistic decomposition:

$$P(w_{1:T}) = \prod_{i=1}^{T} P(w_i|w_{i-1}, ..., w_1) \tag{3}$$

While in practice, LLMs generate new tokens based on all previous tokens, we investigate whether a simplified assumption similar to that of topic models can be made for LLMs:

$$P_M(w_{t+1:T}|w_{1:t}) = \int_{\Theta} P_M(w_{t+1:T}|\theta)P_M(\theta|w_{1:t})d\theta \tag{4}$$

The detailed algorithm is as follows:

```
**Algorithm 1** Latent concept learning
  **Input:** Dataset $\mathcal{D} = \{(x_i, y_i, d_i)\}_i$ associated with a set of tasks $\mathcal{S}$, LLM $M$, number of concept tokens per task $c$, learning rate $\alpha$, and number of training steps $N$.
  **Output:** LLM $M'$ with fine-tuned concept tokens.
  Add $c|\mathcal{S}|$ new tokens to the vocabulary. i.e. The concept tokens $\hat{\theta}^d$ for each task in $\mathcal{S}$. Randomly initialize their embeddings $E_{new}$. Freeze all parameters in $M$ except $E_{new}$;
  **for** step $= 1$ **to** $N$ **do**
    Sample a random batch $B$ in $\mathcal{D}$ and initialize gradient $g \leftarrow 0$;
    **for** each data point $(x, y, d)$ in $B$ **do**
      $g = g + \frac{\partial \ell(X,Y;\hat{\theta}^d)}{\partial E_{new}}$;
    **end for**
    $E_{new} = E_{new} - \alpha g$;
  **end for**
```

Figure 2: Algorithm for Latent Concept Learning

2.2 Demonstration Selection

In this step, our objective is to select demonstrations that can optimally infer the task concept for all test inputs on average. Demonstration selection is crucial, as it directly impacts the model's ability to generalize and perform accurately on unseen tasks. To achieve this, we identify the top k demonstrations from a candidate set, aiming to maximize the likelihood of successfully applying the relevant concept tokens fine-tuned in the previous step. By carefully selecting demonstrations that best represent the task at hand, we can significantly enhance the model's understanding and performance.

Our goal can be mathematically represented by the following equation:

$$\underset{(X_d^1,Y_d^1),\ldots,(X_d^k,Y_d^k)}{\operatorname{argmax}} \left(\mathbb{E}_X \left[P_d^M \left(\theta_d \mid X_d^1, Y_d^1, \ldots, X_d^k, Y_d^k, X\right)\right]\right) \tag{5}$$

To simplify the inherently complex combinatorial search space, we assume independence between demonstrations, allowing us to consider each demonstration separately rather than accounting for all possible combinations. Additionally, given that each task may have multiple concept tokens, these tokens are represented as an ordered sequence based on their increasing token IDs. The detailed algorithm is given below:

Algorithm 2 Demonstration selection

```
Input: dataset $\mathcal{D}^d$ for a task $d$. LLM with fine-tuned concept tokens $M'$. The number of demonstrations $k$.
Output: A set of selected demonstrations.
for each $(X^d, Y^d)$ in $\mathcal{D}^d$ do
    Compute $\hat{P}^d_M(\hat{\theta}^d | X^d, Y^d)$;
end for
Select top $k$ examples with the largest $\hat{P}^d_M(\hat{\theta}^d | X^d, Y^d)$, denoted as $(X^d_1, Y^d_1), ..., (X^d_k, Y^d_k)$;
```

Figure 3: Algorithm for Demonstration Selection

2.3 Few Shot Prompting

In our approach, the prompt structure consists of a sequence of demonstrations, each paired with its instruction and corresponding code snippet. Specifically, we used four of these example pairs *(that is, k = 4)*, which are concatenated sequentially. This sequence of demonstrations is followed by the final instruction that the model needs to process. This structured approach allows the model to draw on the provided examples to generate the appropriate response to the final instruction.

Thus the final structure of our prompt is:

Instruction 1 + Code 1

+

Instruction 2 + Code 2

+

Instruction 3 + Code 3

+

Instruction 4 + Code 4

+

Final Instruction

These demonstration input-output pairs are chosen both the methods: latent concept learning, and random demonstration selection.

3 Experiments

3.1 Datasets Used

We utilize two datasets for evaluating code generation: the HumanEval dataset and the MBPP (Multi-task Benchmark for Programming Problems) dataset. The HumanEval data set consists of 164 programming tasks, each represented by columns including Task ID, Prompt, Canonical Solution, Test, and Entry Point. Each task includes a prompt, a canonical solution implemented in Python, a test case to validate the solution, and an entry-point function name. On the other hand, we utilize the MBPP dataset in its sanitized form, comprising 427 programming prompts for evaluation. The MBPP dataset contains columns such as the source file, task ID, prompt, code, test imports, and test list. This data set provides a diverse set of programming challenges, ranging from basic syntax exercises to complex algorithmic problems. While the HumanEval dataset focuses on generating executable code snippets for specific programming tasks, the MBPP dataset primarily consists of programming prompts accompanied by code solutions and test cases. This distinction in the structure of the dataset allows for a comprehensive evaluation of code generation models across a wide range of problem domains.

3.2 Experimental Settings

We conducted experiments using the Santacoder model, a specialized Large Language Model (LLM) designed specifically for code generation tasks. The prompt configuration parameters were set as follows: Max new tokens = 200 (maximum number of new tokens added to the prompt), Temperature = 0.7 (controls the randomness of token sampling during generation), Num return sequences = 5 (number of generated sequences returned by the model), and Top k = 50 (number of highest probability tokens considered during generation). These parameters were chosen empirically to optimize prompt generation. All computations were performed on the T4 GPU.

3.3 Evaluation Metrics

This section outlines the evaluation metrics used to assess the performance of our model in generating code solutions. These metrics provide quantitative measures to gauge the accuracy, reliability, and similarity of the generated outputs with the golden solution.

The evaluation metrics are defined as follows:

- n: Number of prompts chosen from the data set.
- k: Number of samples of code generated per prompt
- **Pass @ k**: The probability that at least one of the top k code samples generated for a problem passes the compilation and test case tests. The Pass @ k metric is formulated as follows:

$$\text{Pass @ } k = 1 - \prod_{i=1}^{k}(1 - p_i)$$

  where p_i represents the probability of passing the unit tests for the i-th code sample among the top k generated samples.

- **Correctness @ k**: Average correctness in k outputs generated per prompt. Formally,

$$\text{Correctness @ k} = \frac{1}{n}\sum_{i=1}^{n}\frac{\text{Number of 100\% Correct Codes at k outputs per prompt}}{k}$$

- **Similarity @ k**: Average with the Golden Solution at k outputs generated per prompt. Formally,

$$\text{Similarity @ k} = \frac{1}{n}\sum_{i=1}^{n}\text{Average Similarity with Golden Solution at k outputs per prompt}$$

4 Results

In this section, we present the results of our experiments on both the MBPP and Humaneval datasets. Table 1 shows the results for the MBPP dataset, while Table 2 presents the results for the Humaneval dataset.

The results demonstrate that the Latent Concept Demos consistently yield the highest performance metrics among the three demonstration selection methods. This superiority can be attributed to the nature of the Latent Concept Learning algorithm, which enables the model to acquire task-specific knowledge by learning new token embeddings tailored for each task. Unlike Random Demos, which provide examples indiscriminately, Latent Concept Demos are specifically tailored to the task at hand, allowing the model to grasp the underlying concepts more effectively. As a result, the model trained with Latent Concept Demos demonstrates a higher level of understanding and proficiency in generating accurate and functional code.

Table 1: **MBPP Dataset Results**

Parameter	Baseline Result	Latent Concept Demos	Random Demos
Correctness@5	2%	7.2%	1.5%
Correctness@20	0.5%	6.0%	0.3%
Correctness@100	0.3%	5.0%	0.2%
Similarity@5	0.77%	3.0%	0.5%
Similarity@20	0.771%	3.5%	0.4%
Similarity@100	2.70%	7.0%	1.8%
Pass@1	0.6%	4.0%	0.2%
Pass@10	6.07%	11.5%	5.0%
Pass@100	20%	27.0%	15.0%

Table 2: **Humaneval Dataset Results**

Parameter	Baseline Result	Latent Concept Demos	Random Demos
Correctness@5	0.1%	1.2%	0.2%
Correctness@20	0.04%	1.1%	0.03%
Correctness@100	0.008%	1.0%	0.005%
Similarity@5	0.91%	3.5%	0.8%
Similarity@20	0.92%	4.0%	0.7%
Similarity@100	3%	7.5%	2%
Pass@1	0.3%	2.0%	0.4%
Pass@10	4.56%	8.0%	3%
Pass@100	13.2%	18.5%	10%

5 Conclusion

In conclusion, our study demonstrates the effectiveness of In-Context Learning (ICL) in augmenting the performance of the Santacoder model across various code generation tasks. By incorporating task-specific contextual information during model training, ICL significantly improves metrics such as Pass @ k, Correctness @ k and Similarity @ k, underscoring its capability to enhance code generation quality and accuracy. We see that the Latent Concept Demonstrations consistently yield superior results. Overall, our findings underscore the potential of ICL as a valuable technique for refining Language Model-based code generation systems, offering insights into how contextual learning can advance the capabilities of such models.

Moving forward, further exploration is warranted to optimize the implementation of ICL and investigate its applicability across a broader range of code generation tasks. As the field of code generation continues to evolve, the insights gained from this study can inform the development of more sophisticated and effective approaches for generating accurate and functional code.

6 Related Work

1. **Large Language Models Are Latent Variable Models: Explaining and Finding Good Demonstrations for In-Context Learning.** Xinyi Wang, Wanrong Zhu, Michael Saxon, Mark Steyvers, William Yang Wang. arXiv preprint.

2. **Competition-Level Code Generation with AlphaCode.** Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, Thomas Hubert, Peter Choy, Cyprien de Masson d'Autume, Igor Babuschkin, Xinyun Chen, Po-Sen Huang, Johannes Welbl, Sven Gowal, Alexey Cherepanov, James Molloy, Daniel J. Mankowitz, Esme Sutherland Robson, Pushmeet Kohli, Nando de Freitas, Koray Kavukcuoglu, Oriol Vinyals. arXiv preprint.

3. **CodeT: Code Generation with Generated Tests.** Bei Chen, Fengji Zhang, Anh Nguyen, Daoguang Zan, Zeqi Lin, Jian-Guang Lou, Weizhu Chen. arXiv preprint.

4. **Large Language Model-Aware In-Context Learning for Code Generation.** Jia Li, Ge Li, Chongyang Tao, Jia Li, Huangzhao Zhang, Fang Liu, Zhi Jin.

7 Acknowledgments

We would like to express our heartfelt gratitude to Dr. Amar Prakash Azad and Dr. Brij Kumar Chavda from IBM Research Bangalore for providing us with the invaluable opportunity to undertake this project. Their guidance, support, and encouragement have been instrumental in shaping our research endeavors. We are truly thankful for their mentorship and for entrusting us with such an impactful initiative.